Systemunterlagen

Inhaltsverzeichnis

Literatur

1. Günter Schmitt
Mikrocomputertechnik mit dem 16-Bit-Prozessor 8086
Oldenbourg 1986 - MIC Grundlagen und Assembler-Programmierung

2. Hans-Peter Messmer
PC-Hardwarebuch
Addison-Wesley 2000 Hardware - Standardwerk

3. Michael Tischer
PC Intern 5 - Systemprogrammierung
DATA Becker 1995 Standardwerk der Systemprogrammierung

4. Oliver Müller
Assembler - Referenz
Franzis 2000 insbesondere Assembler unter Windows und Einbindung in C / C++

5. E.-W. Dieterich
Assembler - Grundlagen der PC-Programmierung
Oldenbourg 2000 insbesondere Verbindung von Assembler und C / C++

6. Brors Isa
Maschinensprache des IBM-PC/AT
Hüthig, 1992 - Assembler, Systemprogrammierung, nötigste an Hardware

7. Thomas Little
Das PC-Buch
Systhema Verlag, 1992 - Hardware, Systemprogrammierung, wenig Assemblerprogrammierung

8. Wolfgang Link
Assembler Programmierung (Einführung unter MS-DOS)
Franzis-Verlag 1992 - Assembler-Programmierung

9. Holger Schökel
Maschinensprache Einsteiger
DATA Becker 1992 - insbesondere auch TSR- und Treiberprogrammierung

10. H. Weber
Mikrorechnertechnik - Die Intel Mikroprozessorfamilie
R. Oldenburg München Wien 1994 Hardware (nicht nur PC-spezifisch) und Assemblerprogrammierung

11. G. Schmitt
C++ Kurs – technisch orientiert
R.Oldenbourg Verlag München Wien 1999 insbesondere Hardware-Zugriffe unter DOS

12. F. Bollow, K.-P. Köhn
PC-System-Programmierung
VDE-Verlag Berlin Offenbach 1995 in Assembler unter DOS

13. T. Langenkamp
Hardware- und System-Programmierung
te-wi Verlag 1994 Assembler und C, sehr ausführlich VGA-Programmierung

14. BORLAND C++ 3.0
Programmier-Handbuch (u. Benutzer-Handbuch)
Borland 1992 insbesondere auch Inline-Assemblierung

15. Turbo Assembler 3.0
Benutzerhandbuch
Borland 1992 insbesondere Schnittstelle zu Borland C++

16. Frank van Gilluwe
The undocumented PC
Addison-Wesley 1994 alles über BIOS / DOS und Hardware

17. Tom Hogan
Die PC-Referenz für Programmierer
Systhema Verlag 1992 reines Nachschlagewerk

Programmiermodell des INTEL 8086-Prozessors (Real Mode)
enthält die Register mit ihren symbolischen Namen, die für den Programmierer erreichbar sind.

Bit-Nummer	15 … 8	7 … 0	
AX	AH	AL	**Arbeitsregister** allg. Arbeitsregister 16 Bit breit auch als zwei 8-Bit-Register verwendbar H -> High höherwertiges L-> Low niederwertiges Byte
BX	BH	BL	
CX	CH	CL	
DX	DH	DL	
SI			**Offsetregister** Indexregister SI -> Source Index (Quellindex)
DI			DI -> Destination Index (Zielindex)
SP			Stapelzeigerregister SP-> Stackpointer zeigt auf den aktuellen Eintrag im Stack (Stapel)
BP			BP -> Basepointer
CS			**Segmentregister** CS-> Codesegment, zeigt auf Speichersegment mit aktuellem Programmcode
DS			DS->Datensegment, zeigt auf Daten
ES			ES->Extrasegment (2.Datensegment)
SS			SS->Stacksegment, zeigt auf Stapelsegment für Zwischenspeicherung
IP			Befehlszeiger IP-> Instruction Pointer zeigt auf die Speicheradresse mit dem nächsten auszuführenden Befehl
F			Prozessorstatusregister F -> Flagregister , die einzelnen Flags (Bits) weisen auf wichtige interne Prozessorzustände hin.

Flagregister (Statusregister) F :

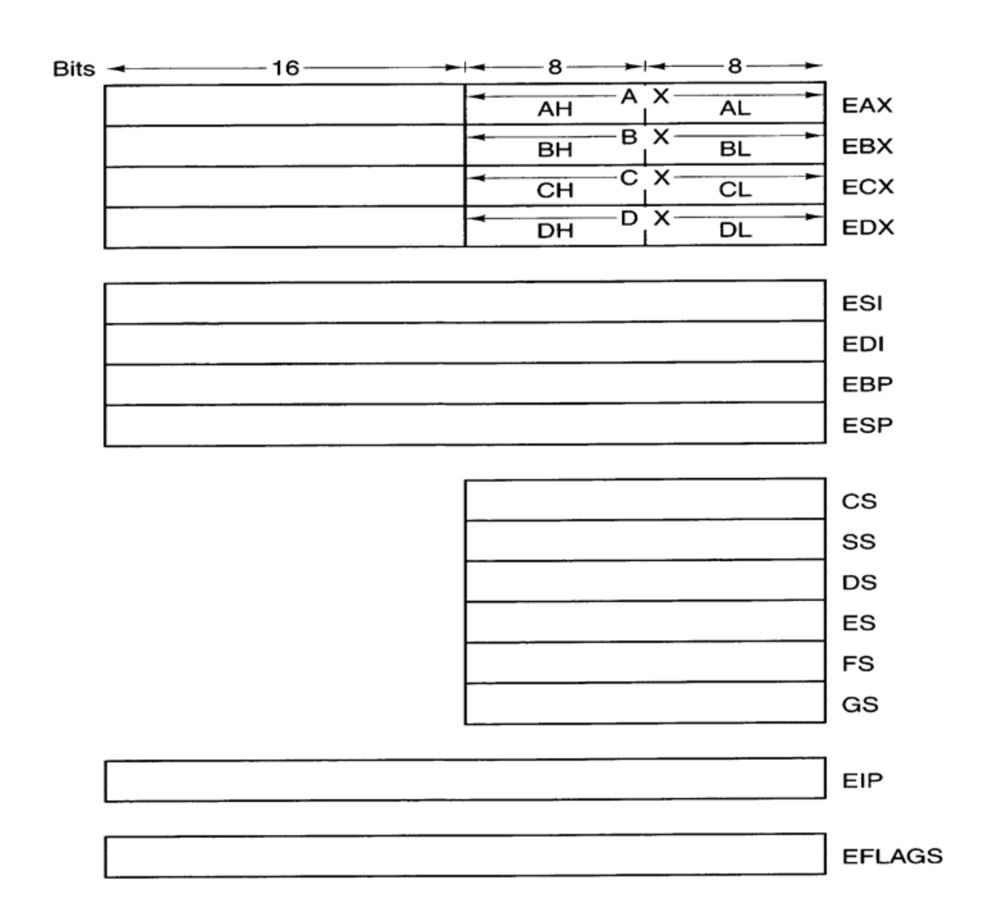
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NT	IO	PL	**O**	**D**	**I**	**T**	**S**	**Z**	0	**A**	0	**P**	1	**C**

Kurzbez.	Bezeichnung	Debug-Bez. (1 / 0)	Bedeutung
C	Carry	CY / NC	C zeigt einen Übertrag aus der höchstwertigen Stelle an wird benötigt bei arithmetischen und logischen Operationen
P	Parity	PE / PO even / odd	Im niederwertigen Byte des Ergebnisses ist die Anzahl der auf 1 stehenden Bits gerade (even) (keine Ergänzung !! nur Anzeige)

A	Auxiliary Carry	AC / NA	zeigt einen Übertrag von Bit 3 nach Bit 4 an benötigt für Dezimalkorrektur beim Rechnen mit BCD-Zahlen
Z	Zero	ZR / NZ	zeigt an, ob das Ergebnis einer Operation 0 ist
S	Sign	NG / PL	zeigt das Vorzeichen eines Ergebnisses an (höchstwertiges Bit)
O	Overflow	OV / NV	Überlauf, d.h. Vorzeichenumkehr, Überschreiten des Zahlenbereiches vorzeichenbehafteter Dualzahlen

Steuerflags

D	Direction	DN / UP	gibt bei Stringoperationen die Indexrichtung (inkrementieren oder dekrementieren) an
I	Interrupt Enable	EI / DI	maskierbare externe Interrupts werden zugelassen (INTR-Eingang) / gesperrt
T	Trap		nach Ausführung eines Maschinenbefehls wird ein Interrupt ausgelöst, um Programme zu testen (Einzelschrittmodus)



Die Register des Pentium II (Register ab 80386)

Zusammenfassung der DEBUG-Befehle

Befehl	Semantik	Format
Assemble	Assembleranweisungen umwandeln	**a** [Adresse]
Compare	Speicherbereiche miteinander vergleichen	**c** Bereich Adr
Dump	Speicherinhalt anzeigen	**d** [Adr] oder d [Bereich]
Enter	Speicherinhalt ändern / eingeben	**e** Adr Liste
Fill	Speicherbereich mit Muster füllen	**f** Bereich Muster
Go	Pgrammausführung mit evtl. Breakpoints	**g** [=Adr][Adrl][Adr2]...
Hexarithm.	hexadez. Addition und Subtraktion	**h** Wert Wert
Input	Byte von I/O-Adresse lesen	**i** Portadresse
Load	Dateien oder abs. Diskettensektoren laden	**l** [Adr [Laufw Sek. Sek.]]
Move	Speicherbereich übertragen	**m** Bereich Adr
Name	Dateien und Parameter definieren	**n** [d:] [Pfad]Name[.erw]
Output	Byte an I/O-Adresse senden	**o** Portadresse Byte
Proceed	Stopp bei nächster Instruktion	**p** [=Adr] [Wert]
Quit	DEBUG-Prog. verlassen, zurück zu DOS	**q**
Register	Register abfragen / Wert zuweisen	**r** [Reg]
Search	Suche nach Zeichen	**s** Bereich Liste
Trace	Ausführung und Protokoll der Register	**t** [=Adr] [Wert]
Unassemble	Code rückübersetzen in Assembleranw.	**u** [Adr] oder u [Bereich]
Write	Dateien oder abs. Diskettensek. schreiben	**w** [Adr [Laufw Sek. Sek.]]

Name in Abhängigkeit des Zustandes des **Flagregisters F** :

Flagname	gesetzt	gelöscht
-Overflow (yes / no)	OV	NV
-Direction (decrement/increment)	DN	UP
-Interrupt (enable/disable)	EI	DI
-Sign (negative/positive)	NG	PL
-Zero (yes / no)	ZR	NZ
-Auxiliary carry (yes / no)	AC	NA
-Parity (yes / no)	PE	PO
-Carry (yes / no)	CY	NC

Softwareentwicklung (mit Borland TASM - TLINK - TD - TLIB)

Der Assembler-Quelltext **name.ASM** soll mit einem ASCII-**Text-Editor** (Text darf nur ASCII-Zeichen enthalten) erstellt werden (z.B. DOS-EDIT oder WORD im Nur-Text-Modus)

Der symbolische Assembler übersetzt den Quelltext in den sogn. Objectcode und erzeugt eine Object-Datei **name.OBJ**
Aufruf : **TASM <name > , , ,** falls keine Endung angegeben wird,sucht der Assembler nach name.ASM

und falls eine Listing-Datei **name.LST** mit der Auflistung von Quell- , Maschinencode und Fehlermeldungen gewünscht wird
Aufruf: **TASM /l <name>**

und falls mit dem Turbodebugger auf Quelltextebene debuggt werden soll
Aufruf: **TASM /zi <name>**

Der Binder (Linker) hat die Aufgabe , eine oder mehrere Objectdateien zusammenzufügen und in die Betriebssystem- umgebung einzubinden und dabei ist die Erstellung einer Reloctionstabelle nötig.
In dieser Tabelle sind alle Positionen im Programm enthalten, deren Inhalt zum Zeitpunkt des Ladens durch DOS an die vorgegebene Speicherplatzierung angepaßt werden müssen.

Aufruf: **TLINK <name_1> + <name_2> , <name>** , **Bibliothek.lib** erstellt aus den beiden Objectdateien

und den benötigten Modulen der Bibiliothek eine **name_1.EXE**-Datei
Zusätzlich zur .EXE-datei wird auch noch eine **.MAP-Datei** mit Kreuzreferenzen erzeugt.

Soll eine **.COM- Datei** erzeugt werden, dann kann dies beim Turbo-Linker durch die Option **/t** erreicht werden und es muß nicht wie beim MASM nach dem normalen Linken die Umwandlung nach COM durch das Programm **EXE2BIN** durchgeführt werden (Umwandlung zu einem ladbaren binärem Programm).

Aufruf: **TLINK /t <name>** erzeugt name.COM -Datei

Mit Option **/v** werden sämtliche **symbolischen** Debugger-Informationen bei der Erstelleung von **EXE-Dateien** mitgelinkt (nicht für .COM-Dateien), damit das Programm mit dem Turbo-Debugger auf der Quellcode-Ebene untersucht werden kann.
Die so entstandenen .EXE oder .COM-Programme können unter DOS durch Eingabe von **name** ausgeführt werden, wobei DOS zuerst nach der name.COM – Datei sucht und dann erst nach der .EXE-Datei.

Für ein komfortables Quelltextdebugging (mit allen Symbolen, im Gegensatz zum DOS-DEBUG)steht der Turbo-Debugger zur Verfügung.
Aufruf: **TD <name>**

Mit **TLIB** lassen sich Bibliotheken aus **.obj-Dateien** bilden, deren Unterprogramme mit **PUBLIC** öffentlich gemacht werden müssen. Im aufrufenden Programm müssen solche externen Programmteile mit **EXTRN** deklariert sein.

TLIB <name> +datei1.obj + datei2.obj , listdat.lst bildet mit den **obj**-Dateien eine name**.lib** – Bibliotheksdatei

Beispiel: **TLIB IOLIB +einaus, IOLIB**
TLINK hauptprog ,,, IOLIB

linkt alle obj.-Dateien der Bibliothek, in denen sich Unterprogramme befinden, die im Hauptprogramm mit EXTRN deklariert wurden , hinzu

Testprogramm mit **.com -Programmrahmen**

```
;noname.asm  Übung Muster für  .com-Programmrahmen mit TASM

escape      EQU     1bh       ; Endemarke definiert

            .MODEL  TINY      ; ein einziges Segment CS=DS=SS=ES
            .486              ; voller Befehlssatz bis 486
            .Code

            ORG     100h
start:      mov     dl,prompt ; DL <- Eingabeprompt
            mov     ah,2      ; Funktion Konsolausgabe
            int     21h       ; DOS
schleife:   mov     ah,1      ; Funktion warten auf Konsole mit Echo
            int     21h       ; DOS AL <= Zeichen von Konsole
            cmp     al,escape ; AL == Endemarke Escape-Taste ?
            je      ausgang   ;   ja: fertig
            jmp     schleife  ; nein: Leseschleife bis Endemarke

ausgang:    mov     ax,3100h  ; zurück nach DOS
            int     21h

prompt      DB      '>'       ; Eingabeprompt
            DB      ' Hier liegen die Testdaten im Datensegment ',0

; hier müssen die benutzerdefinierten internen Unterprogramme liegen

            END      start    ; Ende des Quelltextes
```

Testprogramm mit **.exe - Programmrahmen**

```
;noname.asm  Übung Muster für  exe.-Programmrahmen mit TASM

escape      EQU     1bh       ; Endemarke definiert
            DOSSEG            ; Segmente für DOS
            .MODEL  small     ; 1 Datensegment  1 Codesegment .EXE-Programm
            .486              ; voller Befehlssatz ab 486
            .STACK  256       ; Stapel der Größe 256 Bytes
            .DATA             ; Datensegment
prompt      DB      '>'       ; Eingabeprompt
            DB      ' Hier liegen die Testdaten im Datensegment ',0
            .CODE             ; Codesegment
            .STARTUP          ; Makro: Segmentregister anlegen
            mov     dl,prompt ; DL <- Eingabeprompt
            mov     ah,2      ; Funktion Konsolausgabe
            int     21h       ; DOS
schleife:   mov     ah,1      ; Funktion warten auf Konsole mit Echo
            int     21h       ; DOS AL <= Zeichen von Konsole
            cmp     al,escape ; AL == Endemarke Escape-Taste ?
            je      ausgang   ;   ja: fertig
            jmp     schleife  ; nein: Leseschleife bis Endemarke

ausgang:    .EXIT             ; Makro: zurück nach DOS

; hier müssen die benutzerdefinierten internen Unterprogramme liegen

             END               ; Ende des Quelltextes
```

Unterprogrammbibliothek eados.LIB aus eados.asm

```
; eados.inc enthält die EXTRN Definitionen

; Zuordnung mit INCLUDE eados.inc im aufrufenden Programm
; Aufruf mit CALL name

   EXTRN putch:NEAR        ; AL -> Konsole ausgeben
   EXTRN putblank:NEAR     ; lz ausgeben
   EXTRN putprompt:NEAR     ; cr lf  AL -> Konsole ausgeben
   EXTRN getch:NEAR        ; AL <- Konsole mit warten ohne Echo
   EXTRN getche:NEAR       ; AL <- Konsole mit warten und Echo
   EXTRN puts:NEAR         ; DS:AX = Stringadresse bis 0 ausgeben
   EXTRN gets:NEAR         ; DS:AX = Stringadresse bis cr  lesen
   EXTRN kbhit:NEAR        ; AL == 0: kein Zeichen AL ==-1 abholen
   EXTRN balaus:NEAR       ; AL -> 8 Binärziffern ausgeben
   EXTRN haxein:NEAR       ; AX <= 4 Hexaziffern lesen
   EXTRN halaus:NEAR       ; AL -> 2 Hexaziffern ausgeben
   EXTRN ausnib:NEAR       ; AL -> linkes Nibble ASCII codiert ausg.
   EXTRN haxaus:NEAR       ; AX -> 4 Hexaziffern ausgeben
   EXTRN daxein:NEAR       ; AX <- Dezimalzahl < 65536 lesen
   EXTRN daxaus:NEAR       ; AX -> Dezimalzahl ausgeben
   EXTRN dedez:NEAR        ; AL -> ASCII decodieren C=1: Nicht-Dezi
   EXTRN d32aus:NEAR       ; DX:AX -> dezimal ohne führende Nullen
   EXTRN zufall:NEAR       ; AX=alte -> AX=neue  BX=bereich -> BX=<bereich
   EXTRN wuerfel:NEAR      ; AX <= 40h:6ch Timer-Tic-Variable
   EXTRN taste:NEAR        ; AL == 0: kein Zeichen wie kbhit
   EXTRN grafinit:NEAR     ; Grafikbildschirm ohne Parameter
   EXTRN mausinit:NEAR     ; Maus initialisieren ohne Parameter
   EXTRN lclick:NEAR       ; AX = X-Pos    BX = Y-Pos linke Maustaste
   EXTRN mclose:NEAR       ; Maus schließen ohne Parameter
```

Folgende Möglichkeiten der Benutzung der Unterprogrammbibliothek eados.asm sind gegeben:

1.)

eados.asm mit **TASM eados** (.exe Speichermodell small)

gesamthaft zu einer eados.obj-Objekt-Datei übersetzen und mit

TLINK hauptprog+eados (.exe)

oder TLINK /t hauptprog+eados (.com)

gesamthaft zu einem hauptprog.exe oder .com dazubinden

2.)

oder die UP-Routinen einzeln mit **TASM putch ..** usw. übersetzen, dann die einzelnen Objektdateien mit

TLIB eados.LIB+putch.obj+putblank.obj+ ....,eados.lst

zu einer Bibliothek verbinden.

Mit **TLINK hauptprog,,,eados.LIB**

werden nur die im Hauptprogramm aufgerufenen UP-Routinen dazugelinkt.

Beispielprogramm für Bibliotheksfunktionen

```
; hprueb.asm  Übung externe Uprogramme in eados.LIB bzw. eados.asm
            INCLUDE eados.inc ; Deklarationen (extrn) für eados.LIB
            DOSSEG            ; Segmente für DOS
            .MODEL  small     ; 1 Datensegment  1 Codesegment
            .486              ; voller Befehlssatz ab 486
            .STACK  256       ; Stapelsegment 256 Bytes
            .DATA             ; Datensegment
text        DB  10,13,"Eingabe bis Esc > ",0
ende        DB  10,13,"Auf Wiedersehen > ",0
            .CODE             ; Codesegment
            .STARTUP          ; Makro: Segmentregister anlegen
schlei:     lea     ax,text   ; AX <= String-Anfangs-Adresse
            call    puts      ; Meldung ausgeben
            call    getch     ; AL <= Zeichen lesen ohne Echo
            cmp     al,1bh    ; Ende mit Zeichen Esc ?
            je      mensa     ; ja: Ende der Leseschleife
            cmp     al,'a'    ; nein: untere Grenze ?
            jb      aus       ; kleiner: ausgeben
            cmp     al,'z'    ; obere Grenze ?
            ja      aus       ; grösser: ausgeben
            and     al,11011111b ; maskieren klein -> gross
aus:        call    putch     ; AL Zeichen ausgeben
            call    putblank  ; Leerzeichen
            call    balaus    ; AL binär ausgeben
            call    putblank  ; Leerzeichen
            call    halaus    ; AL hexa ausgeben
            call    putblank  ; Leerzeichen
            xor     ah,ah     ; AH = 0
            call    daxaus    ; AX dezimal ausgeben
            jmp     schlei    ; und weiter bis Esc
mensa:      lea     ax,ende   ; AX <= String-Anfangs-Adresse
            call    puts      ; Text ausgeben
           .EXIT              ; Makro: zurück nach DOS
            END               ; Ende des Quelltextes
```

Bibliothek `eados.LIB` Deklarationen in `eados.INC`
Zuordnung der `EXTRN`– Deklarationen im Quellfile: `INCLUDE eados.inc`

Bezeichner	Funktion	Eingabe	Rückgabe
getch	warten Zeichen ohne Echo lesen	-	AL = Zeichen
getche	warten Zeichen mit Echo lesen	-	AL = Zeichen
putch	Zeichen auf Konsole ausgeben	AL = Zeichen	-
putblank	Leerzeichen (Blank) ausgeben	-	-
gets	String bis Steuerz. < 20h lesen	AX = Pufferadresse	AL = Endezeichen
puts	String bis Endemarke Null ausg.	AX = Pufferadresse	-
halaus	AL in 2 Hexaziffern ausgeben	AL = Byte	-
balaus	AL in 8 Binärziffern ausgeben	AL = Byte	-
daxaus	16 bit in AX dezimal ausgeben	AX = Wert	-

Assembler

Operatoren werden vom Assembler zur Übersetzungszeit ausgewertet und liefern Werte.
Direktiven (Assembleranweisungen) erzeugen keinen Code, sondern steuern die Übersetzung.

Operator / Direktive	**Anwendung**	**Beispiel:**
+ - * /	Vor- und Rechenzeichen	mov ax, tab + 2
:	Segmentvorsatz CS: DS: ES: SS:	mov ax, CS:wert
[ ]	kennzeichnet Adressierungsart	mov ax, [bx] ; indirekt mov ax, DS:[0000] ; direkt
OFFSET	liefert Abstand im Segment	mov bx, OFFSET tab
SEG ausdruck	liefert die Segmentadresse eines Ausdrucks	mov ax, SEG tab
BYTE PTR	Datentyp Byte	inc BYTE PTR [bx]
WORD PTR	Datentyp Wort	inc WORD PTR [si]
NEAR PTR	Sprung oder Aufruf innerhalb des Segments	call NEAR PTR putch
FAR PTR	Sprung / Aufruf außerhalb des Segments (interseg)	call FAR PTR getch
WORD PTR	Sprung indirekt mit Wortadresse (intrasegment)	jmp WORD PTR [di]
anzahl DUP (wert)	Wiederholungsfaktor für Datenvereinbarung	tab DB 10 DUP ('*')
?	nicht vorbesetzte Variable	tab DB 10 DUP (?)
$	laufender Adreßzähler des Assemblers	jmp $ + 10 ; besser Sprungziel
Bezeichner EQU wert	vereinbart Bezeichner für Ausdruck	escape EQU 1B h
ORG ausdruck	legt aktuelle Position im aktuellen Segment fest	ORG 100 h ; Startadresse .COM
END [startadresse]	markiert Ende des Quelltextes	END ; ohne Startadresse END anfang ; mit Startadresse
. DOSSEG DOSSEG	MASM: Anordnung der Segmente für DOS TASM: Anordnung der Segmente für DOS	. DOSSEG DOSSEG
. MODEL typ	tiny small medium compact large huge	. MODEL small
. cpu	Register und Befehle für 286 386 486	. 486
. STACK länge	Grösse des Stapels in Byte	. STACK 256
. DATA	es folgt das Datensegment	. DATA
DB	Speicher für ein Byte (8 Bit)	maxi DB 55 h ; vorbesetzt DB ? ; undefiniert
DW	Speicher für ein Wort (16 Bit)	mini DW 1234 h ; Konstante zeig DW maxi ; Adresse
DD	Speicher für ein Doppelwort (32 Bit)	x DD 100000 ; ganz
DQ	Speicher für ein Vierfachwort (64 Bit)	y DQ 47.11 ; reell
DT	Speicher für ein Zehnerwort (80 Bit)	z DT 1E100 ; reell
. CODE	es folgt das Codesegment	. CODE
. STARTUP	Systemmakro lädt DS SS SP	. STARTUP
. EXIT	Systemmakro zurück nach DOS	: EXIT
SEGMENT [attribut] ENDS	Rahmen für Segment	daten SEGMENT daten ENDS
PROC [attribut] ENDP	Rahmen für Prozedur (Unterprogramm)	putch PROC putch ENDP
MACRO [parameter] ENDM	Rahmen für Makro	aus MACRO ENDM
LOCAL bezeichner	lokaler Bezeichner in PROC bzw. MACRO	LOCAL schleife
PUBLIC: typ liste	Bezeichner auch in anderen Moduln verfügbar	PUBLIC otto : NEAR
EXTRN : typ liste	Bezeichner liegt in anderem Modul	EXTRN otto : NEAR, max : NEAR
INCLUDE	Textdatei in Quelltext einfügen	INCLUDE einaus.inc

Übersicht der 8086-Assemblerbefehle (Maschinenbefehle)

Befehl	Funktion
AAA	ASCII adjust AL for addition
AAD	ASCII adjust for division
AAM	ASCII adjust for multiply
AAS	ASCII adjust for subtraction
ADC Xl ,X2	Add with carry
ADD X1,X2	Addition
AND Xl ,X2	Logical-AND
CALL NAER	CALL im Segment
CALL FAR	CALL über Segmentgrenzen
CBW	Convert byte in word
CLC	Clear carry- flag
CLD	Clear direction flag
CLI	Clear interrupt enable flag
CMC	Complement carry flag
CMP Xl ,X2	Compare
CMPS Xl ,X2	Compare Strings
CWD	Convert word to doubleword
DAA	Decimal adjust AL after addition
DAS	Decimal adjust AL after subtraction
DEC Xl	Decrement
DIV Xl	Unsigned divide
HLT	Halt
IDIV Xl	Signed divide
IMUL Xl	Signed multiply
IN Xl ,X2	Input byte/word from port
INC Xl	Increment Xl um 1
INT 3	Interrupt 3
INT xx	Interrupt xx
INTO	Interrupt on overflow
IRET	Interrupt return
JA cb	Jump short if above (CF=0 and ZF=0)
JAE cb	Jump short if above or equal (CF=0)
JB cb	Jump short if below (CF=l)
JBE cb	Jump short if below or equal (CF=1 or ZF=l)
JC cb	Jump short if carry (CF=1)
JCXZ cb	Jump short if CX register is zero
JE cb	Jump short if equal (ZF=l)
JG cb	Jump short if greater (ZF=0 and SF=OF)
JGE cb	Jump short if greater or equal (SF=OF)
JL cb	Jump short if less (SF/=OF)
JLE cb	Jump short if less or equal (ZF=l or SF/=OF)
JMP cb	Jump short
JMP FAR	Jump far (4-byte address)
JMP NEAR	Jump near
JNA cb	Jump short if not above (CF=l or ZF=l)
JNAE cb	Jump short if not above or equal (CF=l)
JNB cb	Jump short if not below (CF=0)
JNBE cb	Jump short if not below or equal (CF=0 and ZF=0)
JNC cb	Jump short if not carry (CF=0)
JNE cb	Jump short if not equal (ZF=0)
JNG cb	Jump short if not greater (ZF=l or SF/=OF)
JNGE cb	Jump short if not greater or equal (SF/=OF)
JNL cb	Jump short if not less (SF=OF)
JNLE cb	Jump short if not less or equal (ZF=0 and SF=OF)
JNO cb	Jump short if not overflow (OF=0)
JNP cb	Jump short if not parity (PF=0)
JNS cb	Jump short if not sign (SF=0)
JNZ cb	Jump short if not zero (ZF=0)
JO cb	Jump short if overflow (OF=l)
JP cb	Jump short if parity (PF=l)
JPE cb	Jump short if parity even (PF=l)

JPO cb	Jump short if parity odd (PF=0)
JS cb	Jump short if sign (SF=l)
JZ cb	Jump short if zero (ZF=l)
LAHF	Load: AH= flags
LDS Xl ,X2	Load pointer using DS
LEA Xl ,X2	Load effectiv adress
LES Xl ,X2	Load pointer using ES
LODS	Load stringbyte
LODSW	Load stringword
LOOP XX	Loop DEC CX; jump short if CX/=0
LOOPE XX	Loop Equal DEC CX; jump short if CX/=0 and equal (ZF=l)
LOOPNE XX	Loop Not Equal DEC CX; jump short if CX/=0 and not equal
LOOPNZ XX	Loop Not Zero DEC CX; jump short if CX/=0 and ZF=0
LOOPZ XX	Loop Zero DEC CX; jump short if CX/=0 and zero (ZF=l)
MOV Xl ,X2	Move
MOVSB	Move Stringbyte
MOVSW	Move Stringword
MUL	Multipy unsigned
NEG	Negate
NOP	No Operation
NOT	NOT-Vergleich
OR Xl ,X2	Logical-OR
OUT Xl ,X2	Output to port
POP Xl	POP Register from stack
POPF	Pop Flags
PUSH Xl	PUSH Register to stack
PUSHF	PUSH Flags
RCL xx,l	Rotate left carry
RCR xx,l	Rotate right carry
REP (prefix)	Repeat
REPE (prefix)	Repeat equal
REPNE (prefix)	Repeat not equal
REPNZ (prefix)	Repeat not zero
REPZ (prefix)	Repeat zero
RETF	Return far
RET	Return near
ROL xx,1	Rotate left
ROR xx,l	Rotate right
SAHF	Store AH into flags
SAL xx,l	Shift arithmetik left
SAR xx,l	Shift arithmetik right
SBB Xl ,X2	Subtract with borrow
SCAS	Compare Strings
SHL xx,l	Shift left
SHR xx,l	Shift right
STC	Set carry flag
STD	Set direction flag
STI	Set interrupt enable flag
STOS xx	Store String
SUB Xl ,X2	Subtract
TEST Xl ,X2	Test
XCHG Xl ,X2	Exchange
XLAT xx	Translate
XOR Xl ,X2	Exclusive-OR

Interrupt-Übersicht

Interrupt	Beschreibung
BIOS	
00h	DIVIDE ERROR (Division durch 0)
0lh	SINGLE STEP; (80386+), DEBUGGING
02h	NON-MASKABLE INTERRUPT
03h	BREAKPOINT (CPU)
04h	INTO DETECTED OVERFLOW (CPU-Obenauf)
05h	PRINTSCREEN
06h	INVALID OPCODE (80286+)
07h	PROCESSOR EXTENSION, reserviert (80286+)
08h	IRQO, SYSTEM TIMER (80286+)
09h	IRQ1, KEYBOARD DATA READY (80286+)
0Ah	IRQ2, LPT2/EGA,VGA, IRQ9 (80286+)
0Bh	IRQ3, COM2 (80286+)
0Ch	IRQ4, COM1 (80286+)
0Dh	IRQ5, HARDDISK, LPT2 (80286+)
0Eh	IRQ6, DISC CONTROLLER (80286+)
0Fh	IRQ7, PARALLEL PRINTER
10h	GRAPHIC (80286+)
11h	BIOS, GET EQUIPMENT LIST (80486+)
12h	BIOS, GET MEMORY SIZE
13h	LAUFWERKSFUNKTIONEN EIN/AUSGABE
14h	SERIELLE SCHNITTSTELLEN
15h	CASSETTEN RECORDER (8088), erweiterte Auf rufe
16h	KEYBOARD EIN/AUSGABE
17h	PRINTER EIN/AUSGABE
18h	DISKLESS BOOT HOOK (START CASSETTE BASIC), diverse Funktionen, ROM-BASIC
19h	SYSTEM, BOOTSTRAP LOADER
lAh	TIMER EIN/AUSGABE, PCI-I/O
lBh	KEYBOARD, CONTROL-BREAK HANDLER
1Ch	TIMER, SYSTEM TIMER TICK
lDh	SYSTEM DATA, GRAPHIC PARAMETERS
lEh	SYSTEM DATA, DISKETTE PARAMETERS
1Fh	SYSTEM DATA, 8x8 GRAPHICS FONT
DOS	
20h	DOS, TERMINATE PROG RAM
21h	DOS, FUNCTION CALLS
22h	DOS, PROGRAM TERMINATION ADDRESS
23h	DOS, CONTROL-C/CONTROL-BREAK HANDLER
24h	DOS, CRITICAL ERROR HANDLER
25h	DOS, ABSOLUTE DISK READ
26h	DOS, ABSOLUTE DISK WRITE
27h	DOS, TERMINATE AND STAY RESIDENT
28h	DOS, DOS IDLE INTERRUPT
29h	DOS, FAST CONSOLE OUTPUT
2Ah	DOS, CRITICAL ERROR, NETBIOS
2Bh - 2Dh	DOS, RESERVED
2Eh	DOS, PASS COMMAND TO COMMAND INTERPRETER
2Fh	DOS, MUTLIPLEXER
30h	DOS, FAR JMP INSTRUCTION
33h	MOUSE
34h-3Fh	FLOATING POINT EMULATION

Interrupt	**Beschreibung**
Anwender	
40h	DISKETTENFUNKTIONEN, umgeleiteter Interrupt 13
41h	SYSTEM DATA, HARD DISK 0 PARAMETER TABLE
42h	GRAPHIC SERVICES (EGA,VGA), umgeleiteter Interrupt 10
43h	GRAPHIC DATA (EGA,MCGA,VGA)
44h	GRAPHIC EGA-CHARACTERS, NOVELL NETWARE
46h	SYSTEM DATA, HARD DISK 1 DRIVE PARAMETER TABLE
47h-4Eh	Diverse Funktionen
4Fh	SCSI, Common Access Method
50h	Umgeleiteter IRQO
51 h	Umgeleiteter IRQ1
52h	Umgeleiteter IRQ2
53h	Umgeleiteter IRQ3
54h	Umgeleiteter IRQ4
55h	Umgeleiteter IRQ5
56h	Umgeleiteter IRQ6
57h	Umgeleiteter IRQ7
58h	Umgeleiteter IRQ8/0
59h	Umgeleiteter IRQ9/1
5Ah	Umgeleiteter IRQ1O/2
5Bh	Umgeleiteter IRQ1 1/3
5Ch	Umgeleiteter IRQ12/4
5Dh	Umgeleiteter IRQ13/5
5Eh	Umgeleiteter IRQ14/6
5Fh	Diverse Funktionen
60h-66h	Reserviert für spezielle Interrupt-Verarbeitung und diverse Funktionen
67h	LIM EMS, diverse Funktionen
68h-6Ch	Diverse Funktionen
6Dh	On Board VGA
6Eh	Netzwerk API
6Fh	Novell NetWare
70h	IRQ8, CMOS REAL-TIME CLOCK
71h	IRQ9, umgeleitet vom BIOS zu INT0A
72h	IRQ1O, reserviert
73h	IRQ11, reserviert
74h	IRQ12, diverse Funktionen
75h	IRQ13, MATH COPROCESSOR EXCEPTION (286+)
76h	IRQ14, HARD DISK CONTROLLER (286+)
77h	IRQ15, diverse Funktionen
78h	DOS Extender; diverse Funktionen
79h	Diverse Funktionen
7Ah	Novell NetWare, diverse Funktionen
7Bh-7Fh	Diverse Funktionen
80h-F0h	Reserviert für IBM-Basic; diverse Funktionen
F1h-FFh	Reserviert für User-Interrupt

Interrupt - Programmierung mit 80x86

Beispiel: Timer 0 – IRQ0 (System-Uhr Tic im PC) inkrementiert BCD-Zähler (Byte) bei jedem durch Timer 0 bewirkten Einsprung in die zugehörige ISR **(Interruptcontroller ist im PC bereits initialisiert).**

Interruptvektor setzen für jede aktive Interruptanforderung

```
push  es
mov   ax,0
mov   es,ax
mov   es : 20 h , offset  TIMER0 ;   ISR          poke (0x00,  0x20,  FP_OFF( TIMER0) + 0x100 ) ;
mov   es : 22 h, cs                               poke (0x00, 0x22,  FP_SEG( TIMER0) );
pop   es
```

Im PC kann das Setzen des Interrupt-Vektors einfacher durch die **DOS-Funktion 25h INT 21h** erfolgen

```
mov   ah, 25h
mov   al, 08              ; Beispiel einer Interrupt –Vektor-Nummer
mov   dx, offset TIMER0   ; DS:DX   FAR-Zeiger auf  ISR   ->  mov   cx, cs ;    mov   ds, cx
int   21h                 ; da  für .COM-Programme  DS = CS
```

für den Fall, daß ein existierender Interrupt-Vektor auf eine neue ISR verbogen werden und nach Abarbeitung der neuen ISR zur alten ISR verzweigt werden oder vor Beendigung des Programms der alte Vektor wiederhergestellt werden soll, kann der alte Interrupt-Vektor mittels der **DOS-Funktion 35h INT 21h** gelesen und gerettet werden.

```
mov   ax, 3508h
int   21h
mov   INTR08_old, bx       ;gelesenen Intr-Vektor speichern
mov   INTR08_old + 2, es
```

Beispiel einer Wiederherstellung vor Programmende

```
mov   dx, INTR08_old
mov   ax, INTR08_old + 2
mov   ds,ax
mov   ax,2508 h
int   21h
```

selektive Freigabe der aktiven Interruptquelle im PIC 8259 (bzw bei Beendigung des Programms **selektiv sperren**)

```
in    al, 21 h          ; Maskenregister                 outp ( 0x21, ( inp ( 0x21 )  &  0xfd )) ;
and   al, 11111110 b    ; IMR.0 löschen gibt IRQ0 frei
out   21 h, al
```

generelle Freigabe des INTR-Eingangs des 80x86 durch den Befehl **STI** -> I-Flag = 1 **enable ()** ;
(bzw. vor Programmbeendigung die **generelle Sperre** durch **CLI** -> I-Flag = 0) **disable ()** ;
generelle Freigabe des INTR-Eingangs ist im PC-Betrieb grundsätzlich gegeben (Timer, Tastatur, Disk usw.)

Interrupt-Service-Routine ISR für jeden aktiven Interrupt:
- diese ISR muß **<u>alle Register !!!!</u>** unverändert zurückliefern
- falls Unterbrechung dieser ISR möglich sein soll, generelle Freigabe des INTR-Eingang mit STI-Befehl
- vor Verlassen der ISR den Interrupt mit EOI-Befehl bestätigen
- und die ISR mit einem IRET-Befehl beenden

```
void  interrupt  TIMER0 (void);  // Prototyp
void  main  (void )                  // Hauptfunktion
{    ----------    }
```

```
TIMER0   PROC
          push    ax
        ( sti                ; damit  unterbrechbar   )
          mov     al, CS:Zaehler
          inc     al
          daa
          out      CS:Zaehler,al
          mov     al, 20 h     ;
          out     20 h, al     ; unspezifischer  EOI fuer Master
          pop     ax
          iret                 ; return from Interrupt
TIMER0    ENDP
Zaehler   DB      0
```

```
void  interrupt  TIMER0  (void )
  {
   asm
      {  in       al, APORT    // Beispiel fuer
         inc      al           // Port-Ausgabe
         daa                   // beim MVUS 8088
         out      APORT,al  }
    outp  ( 0x20,  0x60 ) ; // spezifischer EOI
  }
```

BIOS-Funktionen

Über die Interrupts 10h bis 1Ah können die verschiedenen Funktionen erreicht werden, die das ROM-BIOS zur grundlegenden Kommunikation zwischen einem Programm und der Hardware zur Verfügung stellt.
Es ist zu beachten , daß die verschiedenen Funktionen des Interrupts 13h getrennt nach ihrem Einsatz in Bezug auf Disketten- und Festplattenlaufwerke in zwei verschiedenen Abschnitten aufgeführt werden.

Übersicht der BIOS-Funktionen

Interrupt 10h Bildschirm

00h Setzen des Video-Modus
01h Definition des Erscheinungsbildes des Cursors
02h Positionierung des Cursors
03h Auslesen der Cursor-Position
04h Auslesen der Lichtstiftposition
05h Auswahl der aktuellen Bildschirmseite
06h Textzeilen nach oben schieben (scrollen)
07h Textzeilen nach unten schieben (scrollen)
08h Auslesen eines Zeichens/Farbe
09h Schreiben eines Zeichens/Farbe
0Ah Schreiben eines Zeichens
0Bh/00h Auswahl der Rahmen-/Hintergrundfarbe
0Bh/01h Auswahl der Farbpalette
0Ch Schreibe Grafikpunkt
0Dh Lese Grafikpunkt
0Eh Schreiben eines Zeichens
0Fh Auslesen des Video-Modus
13h Ausgabe einer Zeichenkette

Interrupt 11h Feststellung der Konfiguration

Interrupt 12h Feststellung der Speichergröße

Interrupt 13h Diskette

00h Reset
01h Status Lesen
02h Lesen
03h Schreiben
04h Verifizieren
05h Formatieren
06h Festplatte
07h Festplatte
08h Format abfragen
09h Festplatte
0Ah Festplatte
0Bh Festplatte
0Ch Festplatte
0Dh Festplatte
0Eh Festplatte
0Fh Festplatte
10h Festplatte
11h Festplatte
12h Festplatte
13h Festplatte
14h Festplatte
15h Feststellung des Laufwerktyps
16h Feststellung eines Diskettenwechsels
17h Diskettenformat festlegen
18h Diskettenformat festlegen

Interrupt 13h Festplatte

00h	Reset
01h	Status lesen
02h	Lesen
03h	Schreiben
04h	Verifizieren
05h	Formatieren
08h	Format erfragen
09h	Anpassung fremder Laufwerke
0Ah	Erweitertes Lesen
0Bh	Erweitertes Schreiben
0Ch	Schreib-/Lesekopf bewegen
0Dh	Reset
0Eh	Controller-Lese-Test (PS/2)
0Fh	Controller-Schreib-Test (PS/2)
10h	Ist das Laufwerk bereit?
11h	Rekalibrieren des Laufwerks
12h	Controller-RAM-Test (PS/2)
13h	Laufwerk-Test (PS/2)
14h	Controller-Diagnose
15h	Feststellung des Laufwerktyps

Interrupt 14h Serielle Schnittstelle

00h	Initialisierung
01h	Zeichen ausgeben
02h	Zeichen einlesen
03h	Status erfragen

Interrupt 15h Alter Kassetten-Interrupt

83h	Flag nach Zeitintervall setzen
84h/00h	Abfrage des Status der Feuerknöpfe der Joysticks
84h/01h	Abfrage der Stellung der Joysticks
85h	SysReq-Taste betätigt
86h	Warten
87h	Speicherbereiche verschieben
88h	Speichergröße über 1 MByte ermitteln
89h	Umschaltung in den Protected Mode

Interrupt 16h Tastatur

00h	Zeichen auslesen
01h	Zeichen vorhanden?
02h	Status der Tastatur erfragen
03h	Wiederholrate einstellen
05h	Tastendruck simulieren
10h	Tastaturabfrage für erweiterte Tastaturen
11h	Tastaturabfrage für erweiterte Tastaturen

Interrupt 17h (paralleler) Drucker

00h	Zeichen ausgeben
01h	Drucker initialisieren
02h	Status des Druckers erfragen

Interrupt 18h ROM-BASIC

Interrupt 19h Booten des Rechners

Interrupt 1Ah Datum und Zeit

00h	Zeit-Zähler auslesen
01h	Zeit-Zähler setzen
02h	Auslesen der Echtzeit-Uhr
03h	Setzen der Echtzeit-Uhr
04h	Auslesen des Datums aus der Echtzeit-Uhr
05h	Setzen des Datums der Echtzeit-Uhr
06h	Alarmzeit setzen
07h	Alarmzeit löschen

BIOS-Interrupt-Routinen für die Tastatur   Aufruf: `int 16h`

Funktion	Eingabe	Rückgabe
AH = 0 Zeichen lesen	-	wartet auf Taste ohne Echo AH = Scan-Code AL = ASCII-Code
AH = 1 Tastatur testen ohne warten !!	-	Ergebnis == 0 (Z = 1): kein Zeichen im Puffer Ergebnis != 0 (Z = 0): Zeichen im Puffer und AX abholen mit Fktn AH = 0
AH = 2 TastaturStatus lesen	-	AL = Status der Modifizierertasten 40:17 h Bit AL.x = 1 -> Taste betätigt bzw Modus ein
AH = 3 Wiederholungsrate und Verzögerung einstellen	AL=5 BL = BH =	Wiederholungsrate: 00h =30 Wps bis 1Fh = 2,0 Wps Verzögerung : 00h=250ms ; 01h=500ms ; 11h = 1 Sek
AH = 5 Tastendruck simulieren durch Schreiben in Tastaturpuffer	CH = CL =	Scancode in den Tastaturpuffer schreiben ASCII-Code AL=0 ok; AL = 1 -> Tastaturpuffer voll
AH = 10h Zeichen lesen von erweiterter Tastatur	-	AH = Scan-Code AL = ASCII-Code entspricht Fktn. 00, unterstützt erweiterte Tastatur
AH =11h Pufferstatus für erweiterte Tastatur	-	Ergebnis == 0 (Z = 1): kein Zeichen im Puffer Ergebnis != 0 (Z = 0): Zeichen im Puffer und AX entspricht Funktion 01
AH =12h TastaturStatus lesen der erweiterten Tastatur	-	AL = 1. Umschaltstatusbyte 40:17 h AH = 2. Umschaltstatusbyte 40:18 h

Aufbau des Statusbytes (40 : 17 h) der **Modifizierertasten**:
<------------ *Zustandstasten(Modus) --- -------------------*|<-----*nur mit anderen Tasten zusammen*------>

Einfg	CapsLock	NumLock	ScrollLock	Alt	Strg	Shft links	Shft rechts

Aufbau des 2. Statusbytes (40 : 18 h) der erweiterten Tastatur
<-- **Tasten gedrückt = 1** -->

Einfüge	CapsLock	NumLock	Scroll	Pause	Sysreq	Alt links	Strg links

40:96   MF2-Tastatur (Tastaturstatusbyte)

Tastatur- ID Abfrage läuft	erstes ID-Zeichen empfangen	Num-Lock- Modus wenn ID = MF2	MF2- Tastatur gefunden	rechte Alt- Taste	rechte Ctrl Taste	Code E0 zuletzt empfangen	Code E1 zuletzt empfangen

40:97   MF2-Tastatur (allgemeiner Tastaturstatus)

Sende- Fehler	Modus LEDs wird aktualisiert	Empfangs- Fehler	Bestätigung ACK von Tastatur empfangen	reserviert (stets 0)	Caps-Lock LED ein	Num-Lock LED ein	Scroll-Lock LED ein

Bit = 1 bedeutet, daß der entsprechende Modus eingeschaltet bzw. die entsprechende Taste gedrückt ist.

Die Tastatur-Statusbyte des BIOS

Wort im Tastaturpuffer	Oberes Byte	Unteres Byte
“Normale“ Taste	Scan-Code	ASCII-Code
Erweiterte Taste	Scan-Code	Null
Eingabe über die “Alt“-Taste	Null	ASCII-Zeichen

Umwandlung eines Make-Code durch das BIOS

Adresse	Größe	Inhalt	Bedeutung
40:1a	Wort	Lesezeiger	zeigt auf das nächste zu lesende Zeichen im Tastaturpuffer
40:1c	Wort	Schreibzeiger	zeigt auf die nächste freie Schreibstelle im Tastaturpuffer
40:1e	32 Byte	Tastaturpuffer	16 Zeichen groß, davon aber nur 15 benutzt
40:80	Wort	Beginn des alternativen Tastaturpuffers	Offset ab Segment 0040 h
40:82	Wort	Ende des alternativen Tastaturpuffers	Offset ab Segment 0040 h

Der vom BIOS verwendete, alternative Puffer kann größer als 32 Byte sein, seine Adresse ist aber auf das Segment 0040h beschränkt

BIOS-Datenbereich des Tastaturpuffers

BIOS- Tastatur-Hooks mit INT 15h

Fkt-Nr.	Bedeutung	Eingabe/Kommentar	Ausgabe/Kommentar
AH = 4F h für Scancode-Manipulation	User Exit des Int 09h mit Scancode in AL Aufruf: INT 15h	AH = 4Fh AL = Scancode CY = 1	Rückgabe CY = 0 Taste ignorieren CY = 1 Taste verarbeiten
AH = 85h für Abfangen von SysReq-Taste	Int 09h ruft Funktion auf wenn Alt+SysReq gedrückt	AH = 85h AL = 0 SysReq gedrückt AL = 1 SysReq losgelassen	AX = 00 CY = 0

BIOS – Drucker-Ausgabe mit INT 17h

Fkt-Nr.	Bedeutung	Eingabe / Kommentar	Ausgabe / Kommentar
AH = 0	Zeichen in AL drucken	DX: log. Drucker 00 = LPT1 (PRN) 01 = LPT2 02 = LPT3 AL: zu druckendes Zeichen	AH = 01, falls kein Druck erfolgte, übrigen Bits wie Status
AH = 1	Druckerschnittstelle initialisieren	DX: wie oben	AH: Status der Schnittstelle nach Rücksprung
AH = 2	Status des Druckers bzw der Schnittstelle ermitteln	DX: wie oben	AH: Statusinformation

Druckerstatus in AH nach Aufruf: AH = 02 INT 17h Bit = 1 : wirksam

Drucker frei (Not Busy)	Bestätigung (Acknowledge)	Kein Papier	Drucker ausgewählt	E / A-Fehler	Nicht verwendet	Nicht verwendet	Time-Out-Fehler

Bildschirmfunktionen von INT 10 h

Funktions Nummer	Bedeutung	Eingabedaten / Kommentar	Ausgabedaten /Kommentar
AH = 00 h	Betriebsart wählen Bildschirm wird hierbei gelöscht	AL = 00h: Text 40 x 25 Monochrom AL = 01h: Text 40 x 25 Color 16 farbig AL = 02h: Text 80 x 25 Monochrom AL = 03h: Text 80 x 25 Color, 16 farbig, 8 AL = 04h: Grafik 320 x 200 Color AL = 05h: Grafik 320 x 200 Monochrom AL = 06h: Grafik 640 x 200 Mono/2 Farben AL = 07h: Monochrom-Bildschirmadapter AL = 12h: Grafik 640x480 16 farbig, 1Seite	keine Bit AL.7 = 1 verhindert beim Aufruf das Löschen des Video-RAMs
AH = 01 h	Darstellungsart (Größe) des Bild-schirm-Cursors festlegen	CH = 0 bis 1F h (b4 ..... b0) Cursor-Anfangszeile CL = 0 bis 1F h (b4 ......b0) Cursor-Endzeile	keine
AH = 02 h	Cursor auf Bildschirm positionieren	DH: Zeilen-Nr. entsprechend Text oder DL: Spalten-Nr. Grafik-Modus BH: Bildschirmseite, im Grafikmode BH=0	keine Die Koordinate (0,0) ist in der linken oberen Bildschirmecke
AH = 03 h	Position des Cursors bestimmen	BH: Bildschirmseite, bei Grafik BH=0	DH: Zeilen-Nr. DL: Spalten Nr. CH=0 bis 0Fh Cursor-Anfangszeile CL=0 bis 0Fh Cursor-Endzeile
AH = 04 h	Position eines Licht-stiftes (Light Pen) mit optischen Sensor ermitteln	keine diese Funktion setzt einen Lichtstift voraus, der am Bildschirmadapter angeschlossen ist	AH=0: Lichtstift nicht aktiv AH=1:Folgende Positionskoordinaten sind gültig DH: Zeilen-Nr. DL: Spalten-Nr. CH: Grafikzeilen-Nr. (0 bis 199) BX: Grafikspalten-Nr.(0 bis 319/639)
AH = 05 h	Auswählen einer Bildschirmseite	AL: Bildschirmseite (0 bis 7, je nach Modus)	keine Bei Grafik wird der gesamte Bild-wiederholspeicher benutzt, daher nur gültig für Textmodi
AH = 06 h	Fenster auf aktueller Seite nach oben rollen (Scrolling Up)	AL: Zahl der Scroll-Zeilen CH: Zeilen-/ CL: Spaltenposition des Scroll-Fensters links oben DH: Zeilen-/Spaltenposition des Scroll- DL: Fensters rechts unten BH: Bildschirmattribut für nachgeschobene Leerzeilen	keine Die Bildschirmattribute sind auf Seite definiert Bildschirm löschen: AL=0 BH=Attribut CH=0 CL=0 DH=24 DL=79
AH = 07 h	Fenster auf aktueller Seite nach unten rol-len (Scrolling Down)	AL: Zahl der Scroll-Zeilen CH: Zeilen-/ CL: Spaltenposition des Scroll-Fensters links oben DH: Zeilen-/Spaltenposition des Scroll-Fensters rechts unten BH: Bildschirmattribut für nachgeschobene Leerzeilen	keine Die Attributsbits sind kompatibel bei Monochrom-/Color-Adaptern
AH = 08 h	Bestimmen eines Zeichens mit Attribut an der Cursorposition	BH: Bildschirmseite (nur bei Textmodi)	AL: gelesenes Zeichen ASCII AH: gelesenes Bildschirmattribut (nur Textmodi)
AH = 09 h	Schreiben von Zeichen mit Attribut an die Cursorposition	AL: zu schreibendes Zeichen ASCII BH: Bildschirmseite (nur Textmodi) BL: Attribut (Textmodi) oder Farbe (Grafikmodi) CX: Wiederholungsfaktor Anzahl Zeichen	keine der Wiederholungsfaktor gilt nur für laufende Zeile (kein Zeilenumbruch) Cursor muß separat bedient werden (Fktn 2) Attribut werden verändert

AH = 0Ah	Schreiben eines Zeichens an aktuelle Cursorposition	AL: zu schreibendes Zeichen ASCII BH: Bildschirmseite (nur Textmodi) BL: ZeichenFarbe (Grafik) CX: Wiederholungsfaktor	keine der Wiederholungsfaktor gilt nur für laufende Zeile (kein Zeilenumbruch) Cursor muß separat bedient werden (Fktn 2) Attribute werden nicht verändert
AH = 0Bh	Farbpalette festlegen	BH: Nummer der Farbpalette (0 .... 7F h) BL: Farbwert für diese Farbpalette (bei CGA nur Modus 4)	Keine ist BH=0 so enthält BL die Hintergrund-/Randfarbe (00 .... 0F h) BH = 01h; BL enthält die Paletten Nr. 0..1
AH = 0Ch	Grafikpunkt schreiben	DX: Zeilen-Nr. Y-Koordinate CX: Spalten-Nr. X-Koordinat AL: Farbe (Bitstelle 0 bis 6) Bit b7 = 1; Exklusiv-ODER-Verknüpfung mit der aktuellen Pixelfarbe BH = 0 Bildschirmseite	Keine über die XOR-Verknüpfung kann eine Bildung der komplementären Farbe erreicht werden
AH = 0Dh	lesen eines Grafikpunktes	DX: Zeilen-Nr. BH = 0 Bildschirmseite CX: Spalten-Nr.	AL: Farbe des gewählten Punktes
AH = 0Eh	Zeichenausgabe mit Weiterpositionierung des Cursors	AL: auszugebendes Zeichen BL: Zeichenfarbe	Keine
AH = 0Fh	Abfrage des aktuellen Bildschirmstatus	keine	AL: Bildschirmmodus (wie Funktion AH = 0) AH: Zahl der Bildschirmspalten (40 o. 80) BH: aktuelle Bildschirmseite
AH = 13h	Ausgabe einer Zeichenkette	AL: Ausgabemodus BL: Attribut-Byte CX: Anzahl Zeichen DX: Zeile-Spalte BH:Bildschirmseite ES:BP Pufferzeiger	Modi: 0=Attribut BL; Cursor beibehalten 1=Attribut BL; Cursor aktualisiert 2=Attribut Puffer; Cursor beibehalten 3=Attribut Puffer, Cursor aktualisiert

Aufbau des Text-Attributes (Mischfarben) auch für den Text-Bildspeicher:

	<------- **Bildfarbe** (Hintergrund) ------->			<---------------------- **Textfarbe** ---------------------->			
Bit 7	rot	grün	blau	hell	rot	grün	blau

Bit 7: Textblinken bzw. Bildhell je nach Treiber

Farbwerte für **Pixelgrafik** mit Helligkeitsbit (für Grafik-Modus 12h 640x480 16 farbig)			
0 = schwarz	4 = rot	8 = dunkelgrau	0CH = hellrot
1 = blau	5 = fuchsin	9 = hellblau	0DH = hellfuchsin
2 = grün	6 = braun	0AH = hellgrün	0EH = gelb
3 = türkis	7 = hellgrau	0Bh = helltürkis	0FH = weiß

Aufbau des Text-Bildspeichers im Segment: **0B800h**

	1. Spalte		**2.Spalte**			**80. Spalte**	
	Code	**Attribut**	**Code**	**Attribut**		**Code**	**Attribut**
1.Zeile	+0000	+0001	+0002	+0003		+0158	+0159
2.Zeile	+0160	+0161	+0162	+0163		+0318	+0319
3.Zeile	+0320	+0321	+0322	+0323		+0478	+0479
24.Zeile	+3680	+3681	+3682	+3683		+3838	+3839
25.Zeile	+3840	+3841	+3842	+3843		+3998	+3999

Interrupt-Routinen für die Maus   Aufruf: `int 33h`

Funktion	Aufruf	Rückgabe
AX = 0 Status lesen	-	AX = Maus vorhanden ? = FFFF h => ja BX = Zahl der Tasten 0000 h => nein
AX = 1 Maus-Zeiger ein	-	Standard (Textmode) -> Softwarezeiger Block
AX = 2 Maus-Zeiger aus	-	-
AX = 3 Mauszustand aktueller Status der Tasten und Position des Mauszeigers	-	BX = Tasten (xxxxx mitte=1 rechte=1 linke=1) CX = X-Position in Pixel DX = Y-Position in Pixel
AX = 4 Mauszeigerposition festlegen	CX = X-Position in Pixel DX = Y-Position in Pixel	
AX = 5 Taste gedrückt? Betätigungen einer Maustaste und letzte Mauszeigerposition	BX = Tastencode 0:linke 1:rechte 2:mitte (Code je nach Treiber)	AX =Tasten (......mitte rechte linke) BX = Anzahl der Tasten-Betätigungen seit letzter Abfrage CX = X-Position in Pixel bei letzter DX = Y-Position in Pixel Betätigung
AX = 6 Taste gelöst? Freigeben einer Maustaste und letzte Mauszeigerposition	BX = Tastencode 0:linke 1:rechte 2:mitte (Code je nach Treiber)	AX =Tasten (......mitte rechte linke) BX = Anzahl der Tasten -Freigaben CX = X-Position in Pixel bei letzter DX = Y-Position in Pixel Freigabe
AX = 7 Horizontalbereich einstellen	CX = linke Grenze DX = rechte Grenze	Fenster für Mauszeiger festlegen z.B. (0..639 Pixel)
AX = 8 Vertikalbereich einstellen	CX = obere Grenze DX = untere Grenze	Fenster für Mauszeiger festlegen z.B. (0..479 Pixel)
AX = 9 Mauszeiger im Grafik-modus definieren	BX = Aktionspunkt horizontal CX = Aktionspunkt vertikal ES: DX= FAR-Zeiger	Wertebereich -16 ... + 16 Adresse der Maskenpuffer Screen-Cursor
AX =0ah Mauszeiger im Text-modus definieren	BX = Mauszeigertyp → CX = Screen-Maske UND DX= Cursor-Maske XOR	0 = Software-Mauszeiger (UND / XOR-Maske) 1 = Hardware-Mauszeiger = Cursor CX/DX= Start/Endzeile Zeiger/Cursor-Blocks
AX =0bh Bewegungszähler der Maus lesen	CX = Zählwert horizontal DX= Zählwert vertikal → seit letztem Aufruf	1 Zählwert (Mickey) = 0,13 mm
AX =0fh Verhältnis Zählwert / Bildschirmpunkt festlegen	CX = horizontales Verhältnis DX= vertikales Verhältnis	Wertebereich 1... 32767 Standard: horizontal = 8 , vertikal = 16
AX =10h Mauszeiger bedingt ausschalten	CX = X-Wert rechte Grenze DX= Y-Wert untere Grenze SI = X-Wert linke Grenze DI = Y-Wert obere Grenze	definiert ein Fenster, innerhalb dessen der Mauszeiger gelöscht wird für erneute Darstellung Funktion 01
AX =13h Schwellenwert für doppelte Geschwindigkeit definieren	DX= Schwellenwert	legt Schwellenwert in Zählwerte (Mickey) /sec fest, ab dem Mauszeiger mit doppelter Geschwindigkeit auf Bildschirm bewegt wird Standardschwellenwert = 64

BIOS-Funktionen für Datum und Uhrzeit im PC (DOS-Systemuhr und Echtzeituhr RTC)

Aufruf	Funktion	Eingabe	Rückgabe
INT 1ah BIOS	Zeitzähler lesen DOS-interne Systemuhr Zahl der Timer-Ticks seit Einschalten des PCs 0:00 Uhr gezählt in **40:6fh - 40:6ch** **40:70h** = 1 > 24-Std-Grenze	AH = 0	AL = 24-Stunden-Angabe CX = Zählerstand high DX = Zählerstand low
INT 1ah BIOS	Zeitzähler setzen DOS-interne Systemuhr Setzen der Timer-Ticks seit 0:00 Uhr	AH = 01 CX = Zähler- high DX= Zähler-low	
INT 1ah BIOS	Uhrzeit lesen – Echtzeituhr	AH = 02	AH = 00 CL = Minute BCD CH = Stunde BCD DH = Sekunde BCD CY = 1 -> Fehler
INT 1ah BIOS	Uhrzeit setzen – Echtzeituhr	AH = 03 CL = Minute BCD CH = Stunde BCD DL = 1 Sommerzeit 0=nein DH = Sekunde BCD	AH = 00 CY = 1 -> Fehler
INT 1ah BIOS	Datum lesen – Echtzeituhr	AH = 04	CL = Jahr BCD CH = Jahrhundert BCD DL = Tag BCD DH = Monat BCD CY = 1 -> Fehler
INT 1ah BIOS	Datum setzen – Echtzeituhr	AH = 05 CL = Jahr BCD CH = Jahrhundert BCD DL = Tag BCD DH = Monat BCD	CY = 1 -> Fehler
INT 1ah BIOS	Alarmzeit setzen – Echtzeituhr bei Erreichen der Alarmzeit wird ein Interrupt 4ah ausgelöst	AH = 06 CL = Minute BCD CH = Stunde BCD DH = Sekunde BCD	CY = 1 -> Fehler
INT 1ah BIOS	Alarmzeit löschen – Echtzeituhr (vor Setzen einer neuen Alarmzeit)	AH = 07	CY = 1 -> Fehler
INT 15h BIOS	Wartezeit setzen nach Ablauf der Wartezeit löst Echtzeituhr Interrupt aus Zeiten in µsec zeitliche Auflösung aber bei 1/1024 Hz Bit 7 des Zielbyte wird nach Ablauf der Wartezeit gesetzt	AH = 83h AL = 00 CX = Zeitintervall high DX = Zeitintervall low BX = Offset des Zielbytes ES = Segment des Zielbyte	AH = 00 AL = Statusregister B CY = 1 -> Fehler
INT 15h BIOS	Wartezeit löschen	AH = 83h AL = 01h	
INT 15h BIOS	Zeitintervall abwarten nach Ablauf des Intervalls wird die Ausführung des aufrufenden Programmes fortgesetzt	AH = 86h CX = Zeitintervall high DX = Zeitintervall low Zeit in µsec	CY = 1 -> Fehler

DOS-Application-Program-Interface

Zu den sogn.DOS-Interrupts zählen die Interrupt-Nr.: 20 h – 2F h . Der wichtigste ist der **INT 21 h – Aufruf einer DOS-Funktion.** Über den Interrupt 21h können mehr als 100 Funktionen erreicht werden, die das DOS einem Programm zur Verfügung stellt und die deshalb als Application-Program-Interface (DOS-API) bezeichnet werden.

Übersicht der Funktionen des Interrupts 21h

Zeicheneingabe

01h Zeicheneingabe mit Ausgabe
03h Empfang eines Zeichens von der seriellen Schnittstelle
06h Direkte Zeichenein-/-ausgabe
07h Direkte Zeicheneingabe ohne Ausgabe
08h Zeicheneingabe ohne Ausgabe
0Ah Eingabe einer Zeichenkette
0Bh Lese Eingabestatus
0Ch Lösche Eingabepuffer und rufe Eingabefunktion auf

Zeichenausgabe

02h Ausgabe eines Zeichens
04h Ausgabe eines Zeichens auf die serielle Schnittstelle
05h Ausgabe auf parallele Schnittstelle
06h Direkte Zeichenein-/-ausgabe
09h Ausgabe einer Zeichenkette

Programmbeendigung

00h Programm beenden
31h Programm beenden, aber im Speicher belassen
4Ch Programm mit Ende-Code beenden

Zugriff auf Unterverzeichnisse

39h Unterverzeichnis erstellen
3Ah Unterverzeichnis löschen
3Bh Aktuelles Verzeichnis setzen
47h Aktuelles Verzeichnis ermitteln

RAM-Speicher-Verwaltung

48h RAM-Speicher reservieren
49h RAM-Speicher freigeben
4Ah Größe eines Speicherbereichs ändern
58h/00h Konzept der Speicherverteilung lesen
58h/01h Konzept der Speicherverteilung setzen
58h/02h Einbindung der UMBs abfragen
58h/03h Einbindung der UMBs festlegen

Zugriff auf Gerätetreiber

44h/00h IOCTL: Lesen des Geräte-Attributs
44h/01h IOCTL: Setzen des Geräte-Attributs
44h/02h IOCTL: Daten von einem Zeichentreiber empfangen
44h/03h IOCTL: Daten an einen Zeichentreiber senden
44h/04h/1 IOCTL: Daten von einem Blocktreiber empfangen
44h/04h/2 DBLSPC: Internen Cache schreiben
44h/04h/3 DBLSPC: Internen Cache schreiben und invalidieren
44h/05h IOCTL: Daten an einen Blocktreiber übertragen
44h/06h IOCTL: Eingabestatus abfragen
44h/07h IOCTL: Ausgabestatus abfragen
44h/08h IOCTL: Ist das Medium wechselbar?
44h/09h IOCTL: Device-Remote-Test
44h/0Ah IOCTL: Handle-Remote-Test
44h/0Bh IOCTL: Zugriffswiederholung setzen
44h/0Ch IOCTL: Kommunikation mit einem Zeichentreiber
44h/0Dh IOCTL: Kommunikation mit einem Blocktreiber
44h/0Eh IOCTL: Letzte Laufwerksbezeichnung ermitteln
44h/0Fh IOCTL: Nächste Laufwerksbezeichnung definieren

44h/10h IOCTL-Unterstützung auf Handle-Ebene abfragen
44h/11h IOCTL-Unterstützung auf Geräte-Ebene abfragen

Uhrzeit und Datum
2Ah Datum abfragen
2Bh Datum setzen
2Ch Uhrzeit abfragen
2Dh Uhrzeit setzen

Diskettenübertragungsbereich
1Ah Setzen der DTA-Adresse
2Fh DTA ermitteln

Directory durchsuchen
11h Suche ersten Directory-Eintrag (FCB)
12h Suche nächsten Directory-Eintrag (FCB)
4Eh Ersten Directory-Eintrag suchen (Handle)
4Fh Nächsten Directory-Eintrag suchen (Handle)

Dateizugriff (FCB)
0Fh Datei öffnen (FCB)
10h Datei schließen (FCB)
13h Datei(en) löschen (FCB)
14h Sequentielles Lesen (FCB)
15h Sequentielles Schreiben (FCB)
16h Erstellen oder Leeren einer Datei (FCB)
17h Datei(en) umbenennen (FCB)
21h Wahlfreies Lesen (FCB)
22h Wahlfreies Schreiben (FCB)
23h Lese Dateigröße (FCB)
24h Setze Datensatznummer
27h Wahlfreies Lesen mehrerer Datensätze (FCB)
28h Wahlfreies Schreiben mehrerer Datensätze (FCB)
29h Dateinamen in FCB übertragen

Dateizugriff (Handle)
3Ch Datei erstellen oder leeren (Handle)
3Dh Datei öffnen (Handle)
3Eh Datei schließen (Handle)
3Fh Datei lesen (Handle)
40h Datei beschreiben (Handle)
41h Datei löschen (Handle)
42h Dateizeiger bewegen (Handle)
45h Handle verdoppeln
46h Handles angleichen
56h Datei umbenennen oder verschieben (Handle)
5Ah Temporäre Datei erstellen (Handle)
5Bh Neue Datei erstellen (Handle)
5Ch/00h Bereich einer Datei gegen Zugriff schützen
5Ch/01h Freigabe eines gesperrten Bereichs in einer Datei
6Ch Erweiterte OPEN-Funktion

Netzwerk-Aufrufe
5Eh/00h Namen des Rechners im Netzwerk ermitteln
5Eh/02h Initialisierungs-String für Netzwerkdrucker festlegen
5Eh/03h Initialisierungs-String für Netzwerkdrucker ermitteln
5Fh/02h Eintrag aus der Netzwerkliste holen
5Fh/03h Eintrag in der Netzwerkliste definieren
5Fh/04h Eintrag aus der Netzwerkliste entfernen

Zugriff auf Interrupt-Vektoren
25h Setze Interrupt-Vektor
35h Inhalt eines Interrupt-Vektors auslesen

Zugriff auf Disketten/Festplatten

0Dh Reset der Blocktreiber
0Eh Auswahl des aktuellen Laufwerks
19h Gerätebezeichnung des aktuellen Laufwerks erfragen
1Bh Informationen über das aktuelle Laufwerk einholen
1Ch Informationen über ein beliebiges Laufwerk einholen
1Fh DPB-Zeiger für das aktuelle Laufwerk ermitteln
32h Zeiger auf DPB für ein beliebiges Laufwerk ermitteln
36h Verbleibende Plattenkapazität ermitteln
53h BPB in DPB umsetzen

Zugriff auf den PSP

26h Erstelle neuen PSP
50h Aktiven PSP setzen
51h Aktiven PSP ermitteln
55h Neuen PSP erstellen
62h Adresse des PSP ermitteln

Zugriff auf DOS-Flags

2Eh Setzen des Verify-Flags
33h/00h Lesen des Break-Flags
33h/01h Setzen des Break-Flags
34h DOS Zeiger auf das INDOS-Flag ermitteln
37h/00h Kennzeichen für Kommandozeilen-Schalter ermitteln
37h/01h Kennzeichen für Kommandozeilen-Schalter setzen
52h Zeiger auf DOS-Info-Block ermitteln
54h Verify-Flag lesen

Zugriff auf Datei-Informationen

43h/00h Attribut einer Datei ermitteln
43h/01h Attribut einer Datei setzen
57h/00h Datum und Uhrzeit der letzten Modifikation einer Datei ermitteln
57h/01h Datum und Uhrzeit der letzten Modifikation einer Datei setzen

Zugriff auf landesspezifische Parameter

38h Landesspezifische Symbole und Formate ermitteln
38h/00h Landesspezifische Symbole und Formate ermitteln
38h/01h Land setzen

Verschiedene Funktionen

30h DOS-Versionsnummer ermitteln
4Bh/00h EXEC: anderes Programm ausführen
4Bh/03h EXEC: anderes Programm als Overlay laden
4Bh/05h EXEC: eigene EXECs anpassen
4Dh Ende-Code ermitteln
59h Erweiterte Fehlerinformationen einholen
60h Dateinamen erweitern
66h/01h Aktuelle Code-Page ermitteln
66h/02h Aktuelle Code-Page festlegen
67h Anzahl der verfügbaren Handles festlegen
68h Dateipuffer leeren

DOS-Interrupt-Routinen für die **Ein- / Ausgabe** von Zeichen und Strings auf der **Konsole**:

Aufruf	**Funktion**	**Aufgabe**
int 21h	AH = 1	Tastatur-Eingabe nach **AL** mit Warten und Echo (Ctrl C -> INT 23h) gefiltert -> Steuerzeichen (cr, lf, bcksp, usw.) funktionieren Erweiterte Tastaturcodes: zuerst 0, erneuter Aufruf -> eigentlicher Code
int 21h	AH = 2	Zeichen aus **DL** auf dem Bildschirm ausgeben ; AL wird verändert gefiltert, dh. Steuerzeichen funktionieren
int 21h	AH = 06 DL= 0 –254 DL= 255	Zeichen-**Eingabe** (ohne Warten ohne Echo) und Zeichen-**Ausgabe** dieses ASCII – Zeichen ausgeben (ASCII-Code 255 nicht möglich) Ein Zeichen von Tastatur **ohne Warten und ohne Echo** nach AL ein Z-Flag = 1 : kein Zeichen bereit; Z-Flag = 0 : Zeichen in AL
int 21h	AH = 7	Tastatur-Eingabe nach **AL** mit Warten ohne Echo (ungefiltert d.h. z.B Ctrl C keine Wirkung)
int 21h	AH = 8	Tastatur-Eingabe nach **AL** mit Warten ohne Echo (Ctrl C -> INT 23h) gefiltert, erweiterte Tastaturcodes: zuerst 0, dann eigentlicher Code
int 21h	AH = 09 DS:DX =	Ausgabe einer Zeichenkette Zeichenkettenende muß '$'-Zeichen sein FAR-Zeiger auf Beginn der Zeichenkette ; AL wird verändert gefiltert, Steuerzeichen funktionieren und Ctrl C -> INT 23h
int 21h	AH = 0Ah DS:DX =	Eingabe einer **Zeichenkette mit Echo** (Ctrl C -> INT 23h) FAR-Zeiger auf Puffer fuer Zeichenkette 1.Pufferbyte muß max. Anzahl der einzugebenden Zeichen enthalten 2.Byte enthaelt Anzahl der tatsächlich eingegeben Zeichen ohne cr Funktion mit cr beenden - cr wird als letztes Zeichen im Puffer eingetragen
int 21h	AH = 0Bh	Tastatur-Test ***ohne*** **Warten** (sofort zurück!!!) AL = 0: keine Taste gedrückt AL = 255: Taste gedrückt, Zeichen im Puffer ! abholen mit AH = 1 oder AH = 8
int 21h	AH = 0Ch AL =	Lösche Tastatur-Eingabepuffer und rufe eine Eingabefunktion auf Nummer der aufzurufenden Funktion (1, 6, 7, 8 , 0Ah) ; AL verändert
int 21h	AH = 3Fh BX =handle CX = DS:DX =	Datei lesen nach Pufferspeicher (bei Tastatur mit Echo) 0 = handle für Tastatur CON Rückgabe: AX = Anzahl gelesener Byte Anzahl der Bytes CY=0 o.k. Adresse des Pufferspeichers CY = 1 Fehler , in AX = Fehlercode (Standard unter DOS ist cooked-Mode = Steuerzeichen z:B. <cr> aktiv)
int 21h	AH = 40h BX =handle CX = DS:DX =	Datei beschreiben aus Pufferspeicher 1 = handle für Bildschirm Rückgabe: AX = Anzahl geschriebener Byte Anzahl der Bytes CY=0 o.k. Adresse des Pufferspeichers CY = 1 Fehler , in AX = Fehlercode

DOS-Funktionen für Interrupt-Vektor Aufruf: INT 21h

Fkt-Nr	Bedeutung	Eingabedaten / Kommentar	Ausgabedaten / Kommentar
25 h	Setzen eines Interrupt-Vektors	AH = 25 h AL = Interrupt-Nummer DS:DX Zeiger Seg :Off auf ISR	keine Ausgabe
35 h	Holen eines Interrupt-Vektors	AH = 35 h AL = Interrupt-Nummer	ES : BX Vektoradresse Segment : Offset der ISR

DOS-Funktionen für Programm-Beendigung Aufruf: INT 21h

Fkt-Nr	Bedeutung	Eingabedaten / Kommentar	Ausgabedaten / Kommentar
31 h	Programm beenden aber im Speicher belassen	AH = 31 h AL = Ende-Code DX = Anzahl reservierende Paragraphen (16 Byte) einschließlich PSP	keine Ausgabe für TSR-Programme (Terminate and stay resident)
4C h	Programm mit Ende-Code beenden	AH = 4C h AL = Ende-Code	Keine Ausgabe Zuverlässigste Beendigung

Undokumentierte Funktionsaufrufe Aufruf: INT 21h

Funktion	Eingabe	Rückgabe
Holen Programmsegment Präfix-Adresse (PSP)	AH = 62 h	BX= Segmentadresse des aktuellen PSP
Setzen Programmsegment-Präfixadresse (PSP)	AH = 50 h BX = Segment-adresse neuer PSP	keine
DOS-Infoblock ermitteln	AH = 52 h	ES:BX = Zeiger auf DOS-Infoblock
Holen des Reentry-Flags **INDOS-Flag**	AH = 43 h	ES:BX Byte-Adresse des INDOS-Flags = 0: keine DOS-Funktion aktiv != 0: DOS-Funktion in Ausführung

Aufbau des DOS-Info-Blocks (DIB)

Offset adresse	Länge / Bytes	Bedeutung
-04h	4	Zeiger auf 1. Speichersteuerblock (MCB)
00h	4	Zeiger auf DOS-Parameterblock
04h	4	Zeiger auf letztbenutzten Puffer
08h	4	Zeiger auf CLOCK-Einheitentreiber
0Ch	4	Zeiger auf CON-Einheitentreiber
10h	2	Maximale Sektorlänge
12h	4	Zeiger auf Puffer
16h	4	Zeiger auf Redirector-Table
1Ah	4	Zeiger auf Dateitabelle

DOS-Funktionen für Datum und Uhrzeit im PC (DOS-interne Systemuhr) INT 21h

Funktion	Eingabe	Rückgabe
Datum abfragen Werte dual zurückgeliefert (Wochentag 0=Sonntag 1= Montag ...)	AH = 2Ah	AL = Tag der Woche CX = Jahr DH = Monat DL = Tag
Datum setzen Werte dual setzen	AH = 2Bh CX = Jahr DH = Monat DL = Tag	AL = 0 -> o.k. AL = 255 -> unplausibel
Uhrzeit abfragen Werte dual	AH = 2Ch	CH = Stunde CL=Minute DH = Sekunde DL = 1/100 Sekunde
Uhrzeit setzen Werte dual	AH = 2Dh CH = Stunde CL = Minute DH = Sek DL = 1/100 Sekunde	AL = 0 -> o.k. AL = 255 -> unplausibel

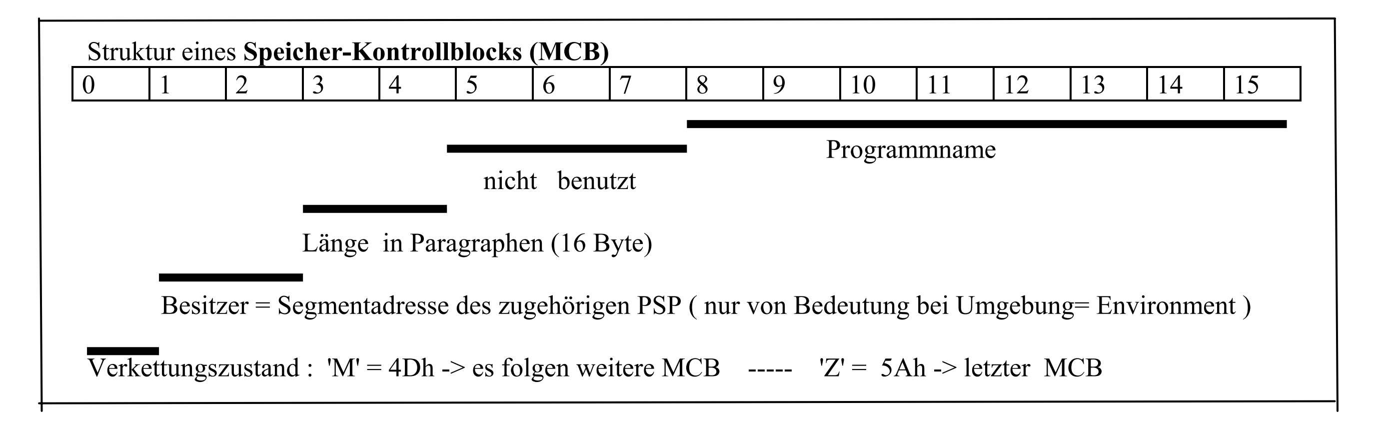
DOS-Funktionen für Verwaltung des Arbeitsspeichers (RAM) Aufruf : INT 21h

Mit dem DOS-Befehl **mem /d |more** läßt sich die Speicherbelegung darstellen

Funktion	Eingabe	Rückgabe
Reservieren von Arbeits-speicher Allocate Memory	AH = 48h BX = benötigte Speichergröße in Paragraphen (16Byte)	CY = 0 erfolgreich AX = Anfangssegment CY = 1 AX = Fehlernummer BX = Größe des größten verfügbaren Speicherblocks
Freigeben von reserviertem Arbeitsspeicher (Free Allocate Memory) – reserviert zuvor mit 48h	AH = 49h ES = Segment des Blockes, der freigegeben werden soll	CY = 0 o.k. CY = 1 AX = Fehlernummer 07->MCBs zerstört
Verändern des reservierten Speicher blocks (SETBLOCK)	AH = 4Ah BX = neue angeforderte Blockgröße in Paragraphen ES = Segment des Speicherblocks	CY = 0 o. k. CY = 1 AX = Fehlernummer BX = Größe des größten verfügbaren Speicherblock
Prüfen / Verändern der Speicher-Reservierungsstrategie	AH = 58 h AL = 0 Prüfen AL = 1 Verändern BX = Strategiecode 00=Tiefenanpas 01=beste Anpassung 02=Hochanpassung	CY = 0 o.k. AX=Strategiecode CY = 1 AX = Fehlernummer 01= ungültiger Funktionscode

Speicherverwaltung unter DOS

erfolgt mit Hilfe von Speicher-Kontrollblocks (**Memory Control Blocks - MCB**) der Größe von 16 Byte, die dem allokierten Speicherbereich im transienten Programm-Bereich (TPA) vorangestellt werden.



Unmittelbar hinter dem Ende des allokierten Speicherbereiches folgt der nächste MCB, damit zeigt die ' Länge ' auch die Entfernung zum nächsten MCB -1 (verkettete Liste) an.
Adresse des ersten MCB im DIB (DOS Information Block). Mit Funktion 52h (undokumentiert) kann die Adresse vom DIB im Registerpaar ES:BX ermittelt werden. Der Pointer (4 Byte) des ersten MCB ist an der Adresse ES: [BX-4] zu finden

Der **Umgebungsblock (Environment)** enthält Strings, die durch die DOS-Befehle **SET** und **PATH** definiert werden (ASCII-String, durch Null-Zeichen beendet)

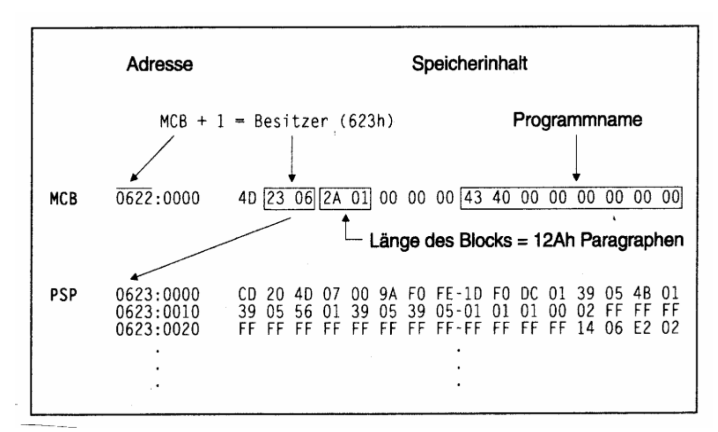
Mit dem DOS-Befehl **mem /d /p** läßt sich die aktuelle Speicherbelegung darstellen.

Ende des freien RAM-Bereiches (TPA)

freier Speicher (durch letzten MCB verwaltet)			
Z	00 00	Länge	
zusätzlicher Speicher (von PROG1 angefordert)			
M	00 00	Länge	
PROG1.EXE			
M	PSP2	Länge	
TEST1.COM			
M	PSP1	Länge	
DOS und BIOS			

00000 h

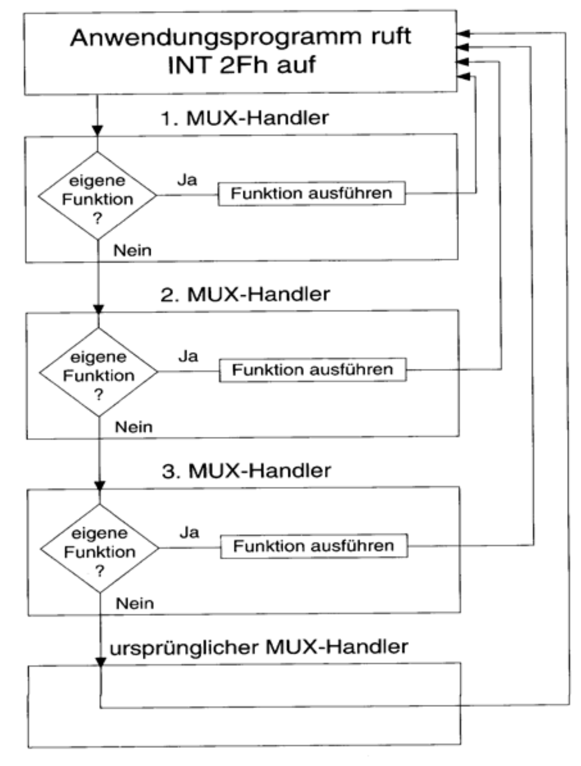
Verwaltung des RAM-Bereiches mit MCBs



Identifikation eines PSP-Blocks im Speicher

Multiplex-Interrupt 2Fh

dient zur Festlegung einer Schnittstelle zwischen Prozessen. Wird für die Verwaltung von DOS-Befehlen, die sich bei ihrem Aufruf als TSR-Programme resident im Speicher installieren (PRINT, ASSIGN, SHARE, APPEND, DOSKEY, ...) verwendet und ermöglicht gewissermaßen bescheidenes Multitasking
Um nach dem resident machen mit diesem Programmen noch in Verbindung treten zu können, klinken sich diese in den **Multiplexer-Interrupt 2F h** ein, der damit für alle TSR-Programme eine Kommunikationsschnittstelle nach außen darstellt um z.B. bestimmte Parameter in dem TSR-Programm zu verändern oder das Programm aus dem Speicher zu entfernen.
Ein Programm das den MUX verwendet, muß sich selbst zunächst eine **8Bit-Kennummer (MUX-Code)** geben, wobei die Nummern 00-BFh für DOS-Programme reserviert sind und der Bereich von C0-FFh für Anwender -programme genutzt werden kann.
Für die Kommunikation mit dem TSR-Programm muß dieses einen eigenen Interrupt-Handler für den MUX 2Fh installieren. Bei einem Aufruf muß diese 2Fh-ISR als erstes durch Vergleich des AH-Registers mit dem MUX-Code feststellen, ob das eigene oder ein anderes Programm gemeint ist.
Bei einem Treffer wird die in den anderen Registern parametrisierte Funktion ausgeführt und das Programm mit einem IRET beendet. Anderenfalls wird zur nächsten MUX-ISR in der Multiplexer-Kette verzweigt.



Kette der MUX-Handler

Bedeutung	Eingabe / Kommentar	Ausgabe / Kommentar
AH: Enthält MUX-Code Aufruf : **INT 2Fh**	AH = 00 – BFh Reserviert von DOS AH = C0h – FFh Frei für Anwendungen	
Rückmeldung Installations-zustand Installationscheck	AL = 00 AH = MUX-Code Prüfung, ob Multiplexer- ISR bereits installiert bzw ob MUX-Code bereits Vergeben	AL = 00 AH = MUX-Code noch nicht installiert AL = MUX-Code AH = 0 bereits installiert
Segmentadresse der MUX-ISR	AL = 01 AH = MUX-Code	AX = Segment

TSR-Anwenderprogramme die den Multiplexer 2Fh nutzen sollten die beiden Unterfunktionen AL = 00 für den **Installationscheck** und AL = 01 für die Rücklieferung der **ISR-Segmentadresse** aufweisen.
Bei 00= Installationscheck werden AH und AL vertauscht zurückgeliefert und damit dem Aufrufer signalisiert, daß eine MUX-ISR mit dem in AH gelieferten MUX-Code bereits installiert ist.
Die zurückgelieferte Segmentadresse der Funktion 01 wird z.B. für die Deinstallation benötigt.

DOS-Funktion für Laden und Ausführen eines Programms (EXEC) Aufruf: INT 21h

EXEC-Funktion 4Bh INT 21h zum Laden und Starten anderer Programme
(wird auch von DOS verwendet= der Kommandoprozessor COMMAND.COM führt durch den Benutzer angegebene Programme mit Hilfe der EXEC-Funktion aus und stellt das Vater-Programm dar)
Vater-Programm ruft Kind-Programm mit 4Bh auf und das Kind-Programm erbt alle Betriebsmittel des Vater-Programms und Rückgabe von Fehlercodes von Kind auf Vater und deren Abfrage mit Funktion 4Dh.
Da .COM- und EXE-Programme den gesamten RAM-Speicher belegen, muß vor dem Ausführen der Kind-Programme der restliche Speicher mit Fktn 4Ah freigemacht werden.

Funktion	Eingabe / Kommentar	Ausgabe / Kommentar
Laden und Ausführen eines Programms alle Register bis auf CS:IP werden verändert zuerst mit 4Ah Arbeitsspeicher teilweise freigeben	**AH = 4B h** ES:BX -> Zeiger auf Übergabeparameterblock DS:DX -> Zeiger auf Dateispezifikation **AL = 00 h** Unterfunktion **Laden und Ausführen** Übergabeparameterblock: W Segmentadresse Umgebungstabelle DW Zeiger auf Befehlsparameter-Zeichenkette (wird kopiert in neuen PSP ab Offset 80 h) DW Zeiger auf ersten FCB (bei PSP + 5Ch) DW Zeiger auf zweiten FCB(bei PSP + 6Ch) **AL = 03 h** Unterfkt **Laden eines Overlays** (ohne PSP) Übergabeparameterblock: W Segmentadresse, Ladeadresse für Datei W Relozierungsparameter für Speicherabbild	CY = 0 alles o.k. CY = 1 AX= codierte Fehlernr. 01: Funktion ungültig 02: Datei nicht vorhanden oder Pfad ungültig 05: Zugriff verweigert 08: Speicher zu klein 0Ah: Umgebungstabelle ungültig 0Bh: Format ungültig

Interrupt 21h, Funktion 4Bh/00h DOS EXEC: anderes Programm ausführen

Diese Funktion ermöglicht es einem Programm, ein anderes Programm ausführen zu lassen, um nach dessen Ausführung selbst weiter abgearbeitet zu werden. Dazu muß der Funktion neben dem Namen des auszuführenden Programms auch die Adresse eines Parameterblocks übergeben werden, der die für die Funktion wichtigen Informationen enthält.

Eingabe	AH =	4Bh
	AL =	00h
	ES:BX =	FAR-Zeiger auf den Parameterblock
	DS:DX =	FAR-Zeiger auf den Puffer mit dem Dateinamen des Programms
Ausgabe	Carry-Flag = 0:	o.k.
	Carry-Flag = 1:	Fehler, in diesem Fall: AX = Fehler-Code
		1: unbekannter Funktionscode
		2: Programm nicht gefunden
		3: Programm nicht gefunden
		4: zu viele Dateien geöffnet
		5: Zugriff verweigert
		8: nicht genügend Speicherbereich
		10: falscher Environment-Block
		11: falsches Format

Der Programmname muß als ASCII-String vorliegen, der durch ein Ende-Zeichen (ASCII-Code 0) abgeschlossen wird. Er darf neben einer Gerätebezeichnung eine komplette Pfadbezeichnung und einen Dateinamen, aber keine Wildcards enthalten. Fehlt die Gerätebezeichnung oder die Pfadbezeichnung, wird auf das aktuelle Gerät bzw. auf das aktuelle Verzeichnis zugegriffen.
Es können nur EXE- oder COM-Programme zur Ausführung gebracht werden. Um eine Batch-Datei auszuführen, muß der Kommandoprozessor (COMMAND. COM) mit dem Parameter /c, gefolgt vom Namen der Batch-Datei, aufgerufen werden.

Der **Parameterblock** muß folgendes Format haben:

Byte 0-1:	Segmentadresse des Environment-Blocks
Byte 2-3:	Offsetadresse der Kommandoparameter
Byte 4-5:	Segmentadresse der Kommandoparameter
Byte 6-7:	Offsetadresse des ersten FCB
Byte 8-9:	Segmentadresse des ersten FCB
Byte 10-11:	Offsetadresse des zweiten FCB
Byte 12-13:	Segmentadresse des zweiten FCB

Wird als Segmentadresse des Environment-Blocks der Wert 0 übergeben, so verfügt das aufgerufene Programm über den gleichen Environment-Block wie das aufrufende Programm.

Die Kommandozeile enthält Befehle für das auszuführende Kind-Progr und wird von DOS in den PSP des Kind-Progr. ab Adresse 80h kopiert. Die Kommandoparameter müssen im Speicher in der Art gespeichert sein, daß zunächst die Anzahl der Zeichen in der Kommandozeile als Byte abgespeichert wird. Darauf folgen die einzelnen ASCII-Zeichen, die durch ein Carriage Return (ASCII-Code 13) beendet werden. Dieses Carriage Return wird allerdings bei der Anzahl der Zeichen nicht mitgezählt.

z.B. PRGPARA DB 9,' /c DIR ', 13

Der erste übergebene FCB wird ab der Adresse 5Ch, der zweite ab der Adresse 6Ch in den PSP des aufgerufenen Programms kopiert. Entnimmt das aufgerufene Programm diesen beiden FCB keine Informationen, können beliebige Werte in die FCB-Felder im Parameterblock eingetragen werden.
Nach dem Aufruf dieser Funktion sind **alle Register bis auf das CS- und das IP-Register zerstört**.**!!!!**
Das aufgerufene Programm verfügt über alle Handles, die auch dem aufrufenden Programm zur Verfügung stehen.

Interrupt 21h, Funktion 4Bh / 03h DOS EXEC: anderes Programm als Overlay laden

Diese Funktion ermöglicht es einem Programm, ein anderes Programm als Overlay in den Speicher zu laden, jedoch ohne daß dieses Programm automatisch ausgeführt wird, d.h. es wird die Kontrolle an das aufrufende Programm zurückgegeben.

Der **Parameterblock** muß folgendes Format haben:

Byte 0-1: **Segmentadresse**, an die das Overlay geladen wird (die Offsetadresse dazu ist 0)
Byte 2-3: **Relokationsfaktor**

Als Relokationsfaktor sollte bei COM-Programmen der Wert 0, bei EXE-Programmen die Segmentadresse angegeben werden, an die das Programm geladen wird.
Da kein PSP mitgeladen wird beginnt deswegen bei .COM-Programmen der Code nicht ab Offset-Adresse 100h sondern ab Adresse 00h relativ zum Ladesegment. Da sich in .COM-Programmen aber bestimmte Sprünge und Datenzugriffe auf Codebeginn 100h beziehen, muß die Startadresse für den FAR CALL-Sprung angepaßt werden:

Ladeadresse Startadresse

CS_{Lade} : 00 = (CS_{Lade} -10h) : 100h = gleiche physikalische Adresse

Um .EXE-Overlays problemlos ausführen zu können, sollte in den zu ladenden EXE-Programmen der erste auszuführende Befehl definiert festgelegt werden (z.B. an den Anfang mit Offset 00h oder auch 100h möglich)
Dann können Ladesegment und Startsegment gleichbleiben.
Nach Laden des Overlays kann das geladene Programm durch einen FAR-Call aufgerufen werden und es gilt:
.COM-Progr.: FAR CALL nach (Ladesegment-10h) : 100h
.EXE-Progr.: FAR CALL nach Ladesegment : Startadresse EXE-Progr.
Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES wird durch diese Funktion nicht verändert.

Interrupt 21h, Funktion 4Bh/05h DOS Eigene EXECs anpassen

Applikationen, die andere Programme oder Overlays unter Umgehung der DOS-Exec-Funktion einladen, müssen sich ab der DOS-Version 5.0 dieser Funktion bedienen, um Probleme beim Nachladen der Programme und Overlays zu vermeiden.

Eingabe AH = 4Bh
AL = 05h
DS:DX = FAR-Zeiger auf die Exec-State-Struktur
Ausgabe Carry-Flag = 0: o.k.
Carry-Flag = 1: Fehler, in diesem Fall: AX = Fehler-Code siehe Funktion 4Bh/ 00h

Der Aufruf dieser Funktion muß zwischen dem Einladen des Programms oder Overlays und dessen Ausführung angesiedelt werden. Zwischen dem Aufruf dieser Funktion und dem Start des Programms bzw. Overlays dürfen dabei weder DOS- oder BIOS-Funktionen, noch irgendwelche anderen Software-Interrupts aufgerufen werden.
In der ExecState-Struktur werden Informationen über das Overlay bzw. Programme zusammengefaßt :

Adr.	Inhalt	Typ
+00h	Reserviert, muß 0 enthalten	1 WORD
+02h	1 = EXE-Programm 2 = Overlay	1 WORD
+04h	Zeiger auf einen ASCII-String mit dem Namen des Programms bzw. Overlays (Pfadangaben sind in diesem String erlaubt)	1 PTR
+08h	Segmentadresse des PSP des Programms bzw. Overlays	1 WORD
+0Ah	Einsprungspunkt in das Programm bzw. Overlay	1 PTR
+0Eh	Programm- bzw. Overlay-Größe inkl. PSP	1 DWORD

Länge: 12h (18 Byte) Tabelle : Aufbau der **ExecState-Struktur**

DOS-Interrupt-Routinen für Dateien Aufruf: `int 21h`

Funktion	Eingabe	Rückgabe
AH = 3DH: alte Datei öffnen Datei *nicht* vorhanden: Cy = 1: Dateifehler	AL = 0: nur lesen AL = 1: nur schreiben AL = 2: lesen und schreiben DS:DX = Stringadresse für Pfad nullterminierter String	AX <= ***handle*** Cy = 0: kein Fehler Cy = 1: Dateifehler AX = Fehlernummer
AH = 3CH: neue Datei öffnen Datei wird immer *neu* erstellt vorhandene Datei wird geleert	CX = Dateiattribut = 0 setzen DS:DX = Stringadresse für Pfad	AX <= ***handle*** Cy = 0: kein Fehler Cy = 1: Dateifehler -> AX
AH = 3FH: geöffnete Datei lesen	BX = ***handle*** CX = Anzahl zu lesender Bytes DS:DX: Pufferadresse der Daten	AX = Zahl der geles. Bytes Cy = 0: kein Fehler Cy = 1: Dateifehler-> AX
AH = 40H: geöffnete Datei schreiben	BX = ***handle*** CX = Anzahl der Bytes DS:DX: Pufferadresse der Daten	AX = Zahl der geschr. Bytes Cy = 0: kein Fehler Cy = 1: Dateifehler
AH = 42H geöffnete Datei positionieren	BX = ***handle*** CX:DX=Anzahl der Positionen AL = 0: ab BOF (Beginn) AL = 1: +/- Ist-Position AL = 2: ab EOF (Ende)	DX:AX= neue Position ab BOF (Beginn) Cy = 0: kein Fehler Cy = 1: Dateifehler AX = Fehlernummer
AH = 3EH: geöffnete Datei schliessen	BX = ***handle***	Cy = 0: kein Fehler Cy = 1: Dateifehler-> AX
AH = 41H: Löschen einer Datei aus einem Verzeichnis	DS:DX = Adresse des Pfadnamens	Cy = 0: kein Fehler Cy = 1: Dateifehler-> AX
AH = 43H: Bestimmen / Setzen Dateiattribut	DS:DX = Adresse des Pfadnamens AL=0 Bestimme Attribute AL=1 Setze Attribute (CX) B3=0	Cy = 0: kein Fehler dann CX= aktuelles Attribut Cy = 1: Dateifehler dann AX = Fehlernummer
AH=56H: Umbenennen einer Datei	DS:DX = Adresse alter Pfadnamen ES:DI = Adresse neuer Pfadnamen	Cy = 0: kein Fehler Cy = 1: Fehler -> AX=Fehlernr

Aufbau des Dateiattributes in CX:

	B2	B1	B0

B0 – 1: Nur-Lese-Datei
B1 – 1: versteckte Datei
B2 – 1: Systemdatei

Die zwingend vorgegebenen Namen der Ein- und Ausgabegeräte

Name	Gerät
CON	Tastatur und Bildschirm
AUX	Serielle Schnittstelle
COM1, COM2	Serielle Schnittstellen
PRN	Parallele Schnittstelle (Drucker)
LPT1, LPT2, LPT3	Parallele Schnittstellen (Drucker)
NUL	Imaginäres Gerät, das Daten verschluckt

Die Standard-Handles

0 Standard-Eingabegerät (CON)
1 Standard-Ausgabegerät (CON)
2 Standardgerät zur Ausgabe von Fehlermeldungen (CON)
3 Serielle Schnittstelle (AUX)
4 Standard-Drucker (PRN)

	HEX	0	1	2	3	4	5	6	7
HEX	BIN	0000	0001	0010	0011	0100	0101	0110	0111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

höherwertiges Halbbyte

niederwertiges Halbbyte

ASCII-Code, 7-Bit-Code nach DIN 66003
(Beispiel: 6 => 0011 0110_2 = 36 h)

Hex-Ziffer 1. → 2. ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		►		0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
-1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
-2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
-3	♥	!!	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
-4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
-5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
-6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
-7	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
-8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
-9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	∙
-A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
-B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
-C	♀	∟	,	<	L	\	l	¦	î	£	¼	╝	╠	▄	∞	ⁿ
-D	♪	↔	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	ϕ	²
-E	♫	▲	.	>	N	^	n	~	Ä	Pt	«	╛	╬	▐	ε	■
-F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	

Erweiterter IBM-PC Zeichensatz für Bildschirmausgabe

Dezimal-Wert	Hex-Code	ASCII-Zeichen	Bedeutung englisch	Bedeutung deutsch
00	00	NUL	Null	Füllzeichen
01	01	S0H	Start of Heading	Anfang des Kopfes
02	02	STX	Start of Text	Anfang des Textes
03	**03**	**ETX**	**End of Text**	**Ende des Textes**
04	04	EOT	End of Transmission	Ende der Übertragung
05	05	ENQ	Enquiry	Stationsaufforderung
06	**06**	**ACK**	**Acknowledge**	**Positive Rückmeldung**
07	07	BEL	Bell	Klingel
08	**08**	**BS**	**Backspace**	**Rückwärtsschritt**
09	09	HT	Horizontal Tabulation	Horizontal-Tabulator
10	**0A**	**LF**	**Line Feed**	**Zeilenvorschub**
11	0B	VT	Vertical Tabulation	Vertikal-Tabulator
12	0C	FF	Form Feed	Formularvorschub
13	**0D**	**CR**	**Carriage Return**	**Wagenrücklauf**
14	0E	SO	Shift Out	Dauerumschaltung
15	0F	SI	Shift In	Rückschaltung
16	10	DLE	Data Link Espace	Datenübertragung-Umschaltung
17	11	DC1	Device Control 1	Gerätesteuerung 1 (**XON**)
18	12	DC2	Device Control 2	Gerätesteuerung 2
19	13	DC3	Device Control 3	Gerätesteuerung 3 (**XOFF**)
20	14	DC4	Device Control 4	Gerätesteuerung 4
21	15	NAK	Negative Acknowledge	Negative Rückmeldung
22	16	SYN	Synchronous Idle	Synchronisierung
23	17	ETB	End of Transmission Block	Ende des Übertragungs-Blocks
24	18	CAN	Cancel	Ungültig machen
25	19	EM	End of Medium	Ende der Aufzeichnung
26	1A	SUB	Substitute	Substitution
27	**1B**	**ESC**	**Escape**	**Umschaltung**
28	1C	FS	File Separator	Hauptgruppen-Trennung
29	1D	GS	Group Separator	Gruppen-Trennung
30	1E	RS	Record Separator	Untergruppen-Trennung
31	1F	US	Unit Separator	Teilgruppen-Trennung
32	**20**	**SP**	**Space**	**Zwischenraum, Leerschritt**
127	7F	DEL	Delete	Löschen

Bedeutung der Sonderzeichen im ASCII-Code nach DIN 66003

Tastencodes (deutsche Tastaturbelegung)

Taste	Scancode		ASCII/erweitert			ASCII/erweitert mit Umschalt			ASCII/erweitert mit Strg			ASCII/erweiter mit Alt		
	Dez	Hex	Dez	Hex	Zeich	Dez	Hex	Zeich	Dez	Hex	Zeich	Dez	Hex	Zeicl
ESC	1	01	27	1b		27	1b		27	1b		1	01	NUL
1 !	2	02	49	31	1	33	21	!				120	78	NUL
2 "	3	03	50	32	2	34	22	"	3	03	NUL	121	79	NUL
3 §	4	04	51	33	3	21	15	o				122	7a	NUL
4 $	5	05	52	34	4	36	24	$				123	7b	NUL
5 %	6	06	53	35	5	37	25	%				124	7c	NUL
6 &	7	07	54	36	6	38	26	&				125	7d	NUL
7 /	8	08	55	37	7	47	2f	/				126	7e	NUL
8 (	9	09	56	38	8	40	28	(				127	7f	NUL
9)	10	0a	57	39	9	41	29	)				128	80	NUL
0 =	11	0b	48	30	0	61	3d	=				129	81	NUL
ß ?	12	0c	225	e1	■	63	3f	?	28	1c		130	82	NUL
' `	13	0d	39	27	'	96	60	`				131	83	NUL
BKSP	14	0e	8	08		8	08		127	7f		14	0e	NUL
TAB	15	0f	9	09		15	0f		148	94	NUL	165	a0	NUL
Q	16	10	113	71	q	81	51	Q	17	11	^Q	16	10	NUL
W	17	11	119	77	w	87	57	W	23	17	^W	17	11	NUL
E	18	12	101	65	e	69	45	E	5	05	^E	18	12	NUL
R	19	13	114	72	r	82	52	R	18	12	^R	19	13	NUL
T	20	14	116	74	t	84	54	T	20	14	^T	20	14	NUL
Z	21	15	122	7a	z	90	5a	Z	26	1a	^Z	44	2c	NUL
U	22	16	117	75	u	85	55	U	21	15	^U	22	16	NUL
I	23	17	105	69	i	73	49	I	9	09	^I	23	17	NUL
O	24	18	111	6f	o	79	4f	O	15	0f	^O	24	18	NUL
P	25	19	112	70	p	80	50	P	16	10	^P	25	19	NUL
Ü	26	1a	129	81	n	154	9a	■	27	1b		26	1a	NUL
+ *	27	1b	43	2b	+	42	2a	*	29	1d		27	1b	NUL
ENTER	28	1c	13	0d		13	0d		10	0a		28	1c	NUL
ENTER[2]	28	1c	13	0d		13	0d		10	0a		28	1c	ERW
Strg li	29	1d												
Strg re[1]	29	1d												
A	30	1e	97	61	a	65	41	A	1	01	^A	30	1e	NUL
S	31	1f	115	73	s	83	53	S	19	13	^S	31	1f	NUL
D	32	20	100	64	d	68	44	D	4	04	^D	32	20	NUL
F	33	21	102	66	f	70	46	F	6	06	^F	33	21	NUL
G	34	22	103	67	g	71	47	G	7	07	^G	34	22	NUL
H	35	23	104	68	h	72	48	H	8	08	^H	35	23	NUL
J	36	24	106	6a	j	74	4a	J	10	0a	^J	36	24	NUL
K	37	25	107	6b	k	75	4b	K	11	0b	^K	37	25	NUL
L	38	26	108	6c	l	76	4c	L	12	0c	^L	38	26	NUL
Ö	39	27	148	94	+	153	99	╓				39	27	NUL
Ä	40	28	132	84	Σ	142	8e	—				40	28	NUL
^ °	41	29	94	5e	^	248	f8	░				41	29	NUL
Umsch li	42	2a												
# '	43	2b	35	23	#	39	27	'				43	2b	NUL
Y	44	2c	121	79	y	89	59	Y	25	19	^Y	21	15	NUL
X	45	2d	120	78	x	88	58	X	24	18	^X	45	2d	NUL

Taste														
C	46	2e	99	63	c	67	43	C	3	03	^C	46	2e	NUL
V	47	2f	118	76	v	86	56	V	22	16	^V	47	2f	NUL
B	48	30	98	62	b	66	42	B	2	02	^B	48	30	NUL
N	49	31	110	6e	n	78	4e	N	14	0e	^N	49	31	NUL
M	50	32	109	6d	m	77	4d	M	13	0d	^M	50	32	NUL
, ;	51	33	44	2c	,	59	3b	;				51	33	NUL
. :	52	34	46	2e	.	58	3a	:				52	34	NUL
- _	53	35	45	2d	-	95	5f	_	31	1f		53	35	NUL
+[2]	53	35	47	2f	/	47	2f	/	149	95	ERW	164	a4	ERW
Umsch re	54	36												
Druck	55	37	42	2a	*	INT 5h4)						SysReq5)		
x[2]	55	37	42	2a	*	42	2a	*	150	96	ERW	55	37	NUL
Alt li	56	38												
Alt re[1]	56	38												
leer	57	39	32	20	SPC	32	20	SPC	32	20	SPC	32	20	SPC
CAPS-Lock	58	3a												
F1	59	3b	59	3b	NUL	84	54	NUL	94	5e	NUL	104	5e	NUL
F2	60	3c	60	3c	NUL	85	55	NUL	95	5f	NUL	105	5f	NUL
F3	61	3d	61	3d	NUL	86	56	NUL	96	60	NUL	106	60	NUL
F4	62	3e	62	3e	NUL	87	57	NUL	97	61	NUL	107	61	NUL
F5	63	3f	63	3f	NUL	88	58	NUL	98	62	NUL	108	62	NUL
F6	64	40	64	40	NUL	89	59	NUL	99	63	NUL	109	63	NUL
F7	65	41	65	41	NUL	90	5a	NUL	100	64	NUL	110	64	NUL
F8	66	42	66	42	NUL	91	5b	NUL	101	65	NUL	111	65	NUL
F9	67	43	67	43	NUL	92	5c	NUL	102	66	NUL	112	66	NUL
F10	68	44	68	44	NUL	93	5d	NUL	103	67	NUL	113	67	NUL
NUM-Lock	69	45												
NUM-Lock[6]	69	45												
Pause[7]	69	45												
Scroll[6]	70	46												
POS 1	71	47	71	47	NUL	55	37	7	119	77	NUL			
POS 1[3]	71	47	71	47	ERW	71	47	ERW	119	77	ERW	151	97	ERW
Cursor ob	72	48	72	48	NUL	56	38	8						
Cursor ob[3]	72	48	72	48	ERW	72	48	ERW	141	8d	ERW	152	98	ERW
Bild ob	73	49	73	49	NUL	57	39	9	132	84	NUL			
Bild ob[3]	73	49	73	49	ERW	73	49	ERW	132	84	ERW	153	99	ERW
-[8]	74	4a	45	2d	-	45	2d	-						
Cursor li	75	4b	75	4b	NUL	52	34	4	115	73	NUL			
Cursor li[3]	75	4b	75	4b	ERW	75	4b	ERW	115	73	ERW	155	9b	ERW
5[8]	76	4c				53	35	5	143	8f	NUL			
Cursor re	77	4d	77	4d	NUL	54	36	6	116	74	NUL			
Cursor re[3]	77	4d	77	4d	ERW	77	4d	ERW	116	74	ERW	157	9d	ERW
+[8]	78	4e	43	2b	+	43	2b	+	144	90	NUL	78	4e	NUL
Ende	79	4f	79	4f	NUL	49	31	1	117	75	NUL			
Ende[3]	79	4f	79	4f	ERW	79	4f	ERW	117	75	ERW	159	9f	ERW
Cursor un	80	50	80	50	NUL	50	32	2	145	91	NUL			
Cursor un[3]	80	50	80	50	ERW	80	50	ERW	145	91	ERW	160	a0	ERW
Bild un	81	51	81	51	NUL	51	33	3	118	76	NUL			
Bild un[3]	81	51	81	51	ERW	81	51	ERW	118	76	ERW	161	a1	ERW
Einfg[3]	82	52	82	52	NUL	48	30	0	146	92	NUL			
Einfg	82	52	82	52	ERW	82	52	ERW	146	92	ERW	162	a2	ERW
Entf[3]	83	53	83	53	NUL	44	2c	,	147	93	NUL			
Entf	83	53	83	53	ERW	83	53	ERW	147	93	ERW	163	a3	ERW
< >[1]	86	56	60	3c	<	62	62	>						
F11	87	57	133	85	ERW	135	87	ERW	137	89	ERW	139	8b	ERW
F12	88	58	134	86	ERW	136	88	ERW	138	8a	ERW	140	8c	ERW

[1] alphanumerischer Block; nur MF II mit Vorsatzbyte 0eh
[2] numerischer Block; nur MF II mit Vorsatzbyte 0eh
[3] im abgesetzten Steuerblock bei MF II
[4] Bildschirmausdruck über INT 05h unter DOS
[5] SysReq entsprechend INT 15h, Funktion 85h
[6] MF II mit Vorsatzbyte e0h
[7] nur MF II mit Vorsatzbyte e1h
[8] numerischer Block

Tastencodes (deutsche Tastaturbelegung)

Die parallele Schnittstelle PIO 8255 (MVUS 80535)

Nach einem Reset sind alle Ports als Eingang geschaltet; die Datenregister sind gelöscht.

A-Port Ausgabe: LED Eingabe: Kippschalter
B-Port Ausgabe: LED Eingabe: Kippschalter
C-Port Ausgabe: LED Eingabe: Kippschalter und Taster
Steuerport nur beschreibbar, nicht rücklesbar

Steuerbyte der Betriebsart 0 (einfache Ein-/Ausgabe):

```
1  0  0  x  x  0  x  x
                     0: Cl-Port = aus
                     1: Cl-Port = ein
                  0: B-Port = aus
                  1: B-Port = ein
            0: Ch-Port = aus
            1: Ch-Port = ein
         0: A-Port = aus
         1: A-Port = ein
```

A-Port	B-Port	Ch-Port	Cl-Port	Steuerbyte
Ausgang	Ausgang	Ausgang	Ausgang	80H
Ausgang	Ausgang	Ausgang	Eingang	81H
Ausgang	Ausgang	Eingang	Ausgang	88H
Ausgang	Ausgang	Eingang	Eingang	89H
Ausgang	Eingang	Ausgang	Ausgang	82H
Ausgang	Eingang	Ausgang	Eingang	83H
Ausgang	Eingang	Eingang	Ausgang	8AH
Ausgang	Eingang	Eingang	Eingang	8BH
Eingang	Ausgang	Ausgang	Ausgang	90H
Eingang	Ausgang	Ausgang	Eingang	91H
Eingang	Ausgang	Eingang	Ausgang	98H
Eingang	Ausgang	Eingang	Eingang	99H
Eingang	Eingang	Ausgang	Ausgang	92H
Eingang	Eingang	Ausgang	Eingang	93H
Eingang	Eingang	Eingang	Ausgang	9AH
Eingang	Eingang	Eingang	Eingang	9BH

Der Interruptsteuerbaustein PIC 8259 im PC

I7	I6	I5	I4	I3	I2	I1	I0

Port 21 h Master Maskenregister (Interruptfreigabe) 0 = frei 1 = gesperrt
Port A1 h Slave

0	1	1	0	0	x	x	x

Port 20 h Master Bestätigungsregister IRQ0: xxx = 000 bis IRQ7: xxx = 111
Port A0 h Slave

Die beiden Interruptsteuerbaustein PIC 8259 A (Master und Slave) werden beim Neustart durch das Betriebssystem durch 4 ICWS (Initialisierungsworte) initialisiert auf:
- steigende Flanke **Flankentriggerung** *mit Bit LTIM = 0 im ICW1*
- zwei kaskadierte PICs **Master-Slave** *mit Bit SNGL = 0 im ICW1*
- 8086-Modus -> Controller legt **Vektor-Nummer (Byte)** auf Datenbus *mit Bit µPM = 1 im ICW4*
  Master -> Vektor-Nr. 08 h (IRQ0) bis 0F h (IRQ7) und *mit Bits Off7-Off3 im ICW2*
  Slave -> Vektor-Nr. 70 h (IRQ0) bis 77 h (IRQ7)
- festliegende **hierarchische Priorität:** Eingang 0 = IRQ0 – höchste Eingang 7= IRQ7 – niedrigste
- normaler **Nestet-Modus**-> verschachtelte Interrupt-Bearbeitung möglich, d.h. ein Interrupt höherer Priorität kann eine laufende Interruptbearbeitung niedriger Priorität unterbrechen. *mit Bit SFNM = 0 im ICW4*
- **manueller EOI** (End of Interrupt)- Befehl signalisiert der CPU eine beendete Interruptbearbeitung. *mit Bit AEOI = 0 im ICW4*

ein **unspezifischer EOI-Befehl** setzt höchst priorisiertes ISR-Bit zurück
(Schreiben EOI-Befehls (20h) nach Port 20h/A0h , durch hierarchische Prioritäten im PC üblich)
ein **spezifischer EOI-Befehl** (6xh) setzt das angegebene Bit im ISR zurück

Im Betrieb muß jeder Interrupteingang durch Schreiben eines OCW1 in das Maskenregister (Port 21h/A1h) freigegeben (0) bzw. gesperrt (1) werden. Am Ende eines Interruptprogramms (Interrupt-Service-Routine) ist die Annahme im Bestätigungsregister (Port 20h/A0h) durch Schreiben eines OCW2 mit EOI-Befehl zu bestätigen..

Adressen und Steuerbytes dieser Betriebsart im PC mit spezifischem EOI

Gerät	Interr.	Freigabe	Sperre	EOI-Best.	Vektor	Adresse
Timer 0	IRQ0	AND 0FEH	OR 01H	60H	08H	0:0020H
Tastatur	IRQ1	AND 0FDH	OR 02H	61H	09H	0:0024H
PIC2-Slave	IRQ2	AND 0FBH	OR 04H	62H	0AH	0:0028H
COM 2	IRQ3	AND 0F7H	OR 08H	63H	0BH	0:002CH
COM 1	IRQ4	AND 0EFH	OR 10H	64H	0CH	0:0030H
LPT2	IRQ5	AND 0DFH	OR 20H	65H	0DH	0:0034H
Disk	IRQ6	AND 0BFH	OR 40H	66H	0EH	0:0038H
LPT1	IRQ7	AND 07FH	OR 80H	67H	0FH	0:003CH

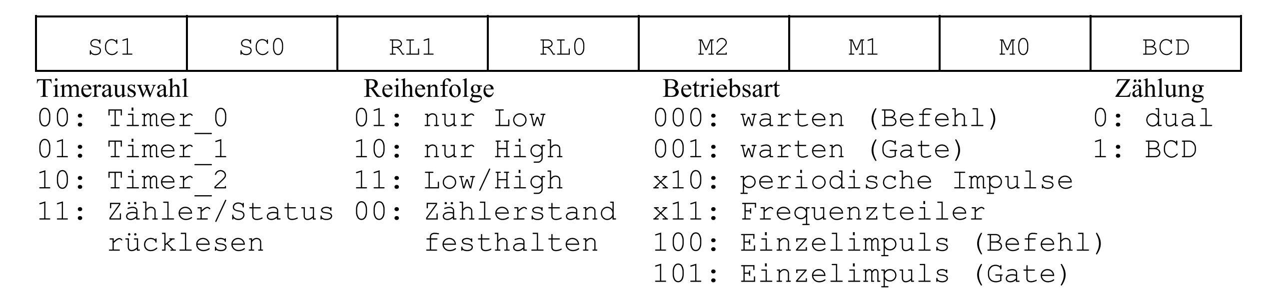
Der Interruptsteuerbaustein 8259A (Master , voreingestellt)

Beispiel: Zurücksetzen der ISR-Bit nach Bedienung des IRQ13 – Coprozessor durch unspezifischen EOI-Befehl 20 h

```
MOV   AL, 20 h     ; EOI-Befehl
OUT   0A0 h, AL    ; an Slave
OUT   20 h , AL    ; an Master
```

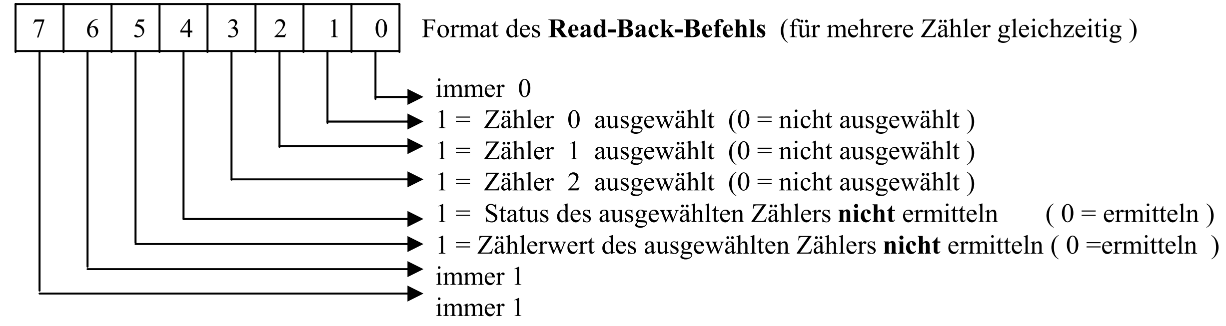
Der Timerbaustein PIT 8254

Die Anschlüsse `Clk`, `Gate` und `Out` sind im PC mit dem Takt und der Gate-Steuerung (+5 V) fest verbunden.
Aufbau des Steuerbytes für den Steuerport 43h:



SC1	SC0	RL1	RL0	M2	M1	M0	BCD

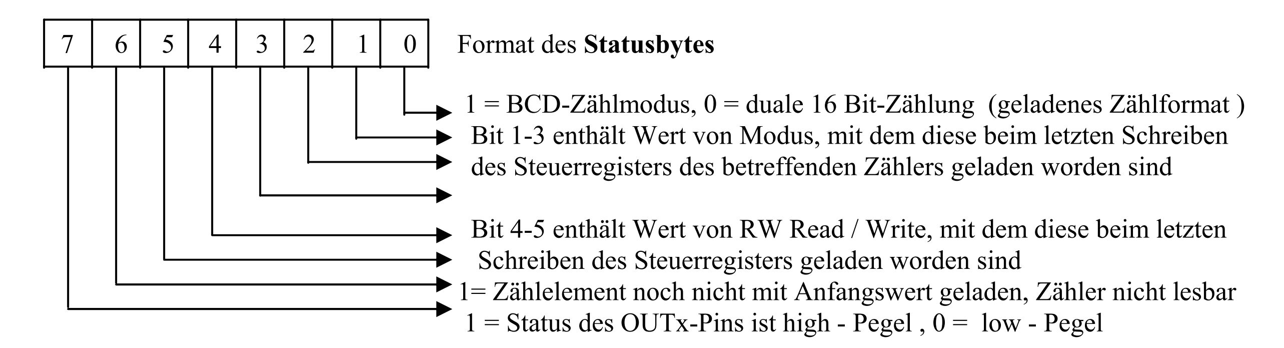
```
Timerauswahl           Reihenfolge         Betriebsart                   Zählung
00: Timer_0            01: nur Low         000: warten (Befehl)          0: dual
01: Timer_1            10: nur High        001: warten (Gate)            1: BCD
10: Timer_2            11: Low/High        x10: periodische Impulse
11: Zähler/Status      00: Zählerstand     x11: Frequenzteiler
    rücklesen              festhalten      100: Einzelimpuls (Befehl)
                                           101: Einzelimpuls (Gate)
```



Beispiel:

Modus des Zählers 0 ermitteln

```
MOV   AL,11100010b      ;Read-Back nur Modus
OUT   43h,AL            ;an Steuerregister
IN    AL,40h            ;Modus für Timer 0 ein
```



Im obigen **Beispiel** könnte im PC das zurückgelieferte Modusbyte wie folgt aussehen:

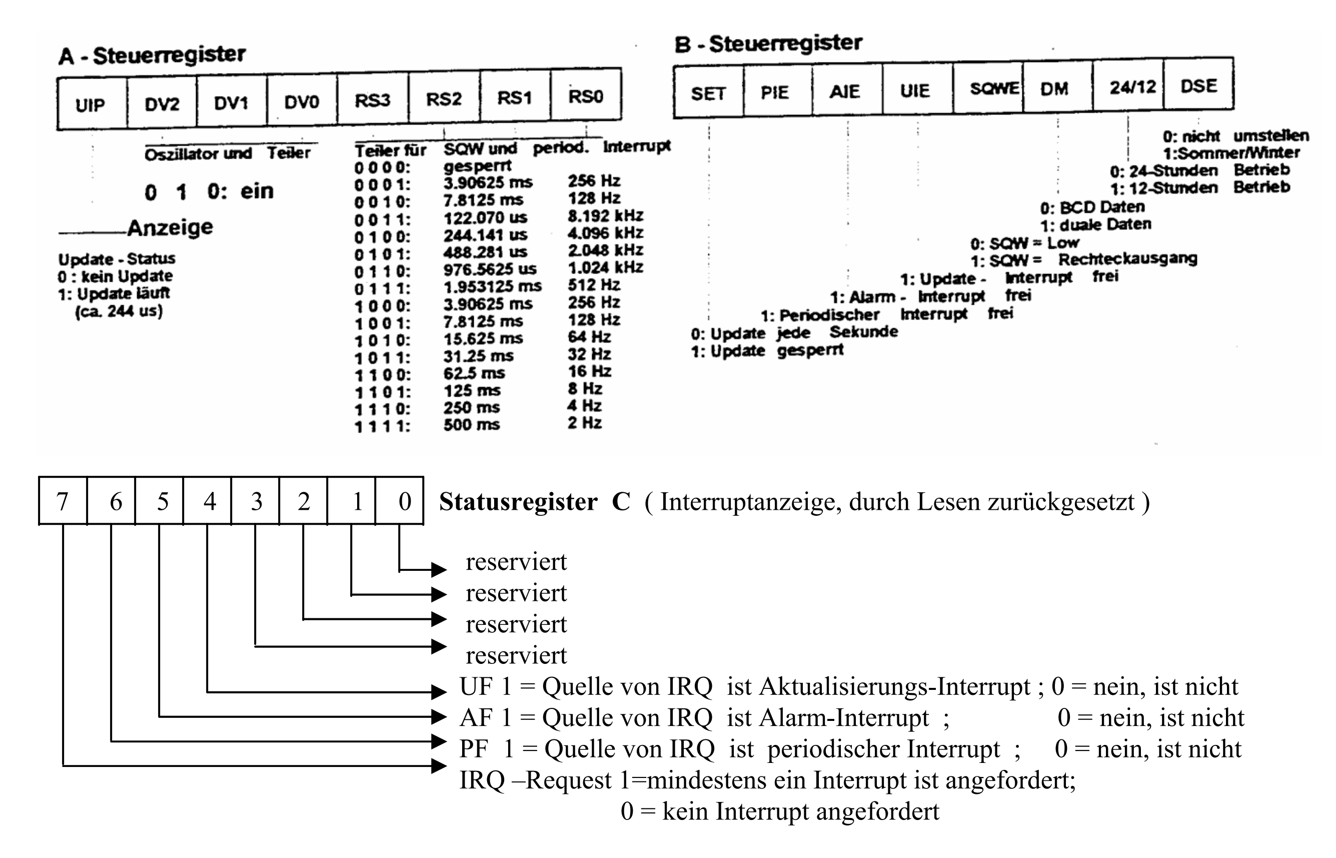
Statusregister: 0 0 1 1 0 1 0 0 (INT 08h Systemuhr)

-> OUT-Pin auf 0V
-> Anfangszählwert geladen
--> Lesen / Schreiben beider Byte
---> Zählermodus 2 (per. Ratengenerator)
-→ duale Zählung

Der Uhrenbaustein RTC

Byte	Adresse	Inhalt	Byte	Adresse	Inhalt
0	00h	Sekunde*)	22	16h	Basisspeicher (höherw. Byte)
1	01h	Alarmsekunde*)	23	17h	Extended Memory (niederwert.
2	02h	Minute*)			Byte) laut SETUP
3	03h	Alarmminute*)	24	18h	Extended Memory (höherwert.
4	04h	Stunde*)			Byte) laut SETUP
5	05h	Alarmstunde*)	25	19h	Erweiterungsbyte 1. Festplatte
6	06h	Tag der Woche*)	26	1ah	Erweiterungsbyte 2.Festplatte
7	07h	Tag des Monats*)	27-31	1bh-1fh	reserviert
8	08h	Monat*)	32-39	20h-27h	Parameter Festplattentyp 48
9	09h	Jahr*)	40-45	28h-2dh	reserviert
10	0ah	Statusregister A	46	2eh	Prüfsumme (niederw. Byte)
11	0bh	Statusregister B	47	2fh	Prüfsumme (höherw. Byte)
12	0ch	Statusregister C	48	30h	Extended Memory (niederwert.
13	0dh	Statusregister D			Byte) laut POST
14	0eh	Diagnosestatus	49	31h	Extended Memory (höherwert.
15	0fh	Shutdown-Status			Byte) laut POST
16	10h	Typ der Diskettenlaufwerke	50	32h	Jahrhundert*)
17	11h	reserviert	51	33h	Setup-Informationen
18	12h	Typ der Festplattenlaufwerke	52	34h	reserviert
19	13h	reserviert	53-60	35h-3ch	Parameter Festplattentyp 49
20	14h	Gerätebyte	61-63	3dh-3fh	reserviert
21	15h	Basisspeicher (niederw. Byte)			

*) üblicherweise binärkodierte Dezimalzahlen (1 Byte)



UF, AF und PF werden durch die jeweilige Aktivität gesetzt (für Polling-Abfrage), und dies ist nicht abhängig von der Interrupt-Freigabe im Statusregister B